図解 マイクロコンピュータ

64180の使い方

Ph.D. 石川 知雄 編 ● オーム社

HC 137×1/2
HC 157×3
HC 74
VUP
+5V
+5V
IOE
HC 74
4.7k ×2
100
1 500 P
ACK
STB
4.7k×8

図解 マイクロコンピュータ

64180の使い方

● Ph.D. 石川知雄 編

オーム社

執筆者一覧

石川　知雄　株式会社日立製作所
馬場　志郎　株式会社日立製作所
川下　智恵　株式会社日立製作所
越路　誠一　株式会社日立製作所
北川　信男　株式会社日立製作所
高野　　誠　日立マイクロコンピュータエンジニアリング株式会社
井戸　慎一　日立マイクロコンピュータエンジニアリング株式会社
星　　恭彦　日立マイクロコンピュータエンジニアリング株式会社

推薦のことば

過　去

1970年度初頭のマイコンの登場は，世の中に大きな変化をもたらしました．以前は夢であった真のパーソナルコンピュータが現実のものとなったのです．

その頃，既存の大きなコンピュータ・メーカはデスクトップ・コンピューティングなる概念には簡単にはなじめなかったのです．このことは将来の夢に駆られた数多くの小さなハードウェア，ソフトウェアの会社に対し，大きなオープン・マーケットを提供することになったのです．

私の最初のパソコンは8080 ACPUボード（2MHz）と4Kバイト RAMボードをのせたIMSAIのボックスでした．プログラミングはバイナリコードを前面のパネルスイッチから入力することで行い，アプリケーションも入出力の8個のスイッチと8個のLED表示で限定されたものでした．

このようなみすぼらしい始まりでしたが，新しい技術により性能が急速に向上するまでにはあまり長い時間を要しませんでした．強化されたZ80プロセッサ，フロッピーディスク，集積度の高いRAM，低価格のビデオ・ターミナルの登場が，CP/M（Digital Research），MBASIC（Microsoft），Wordstar（MicroPro），dBASE（Ashton-Tate）などのパイオニアとなったソフトウェア製品に，格好の土台を提供したのです．これらの，アセンブリ言語で書かれ，人手で高速化，効率化のためチューン・アップされたプログラムは，その後のテスト期間に耐えたのです．事実これらのプログラムは現在の水準に対しても高度のプログラム芸術といえましょう．

その後も技術の進歩は続き，1970年代の末にはインテル8088/8086，モトローラ68000，ZilogZ8000の16ビットの戦いが焦点となりました．新しいマイコンに人材が集中し，8ビットの市場は放棄されたかに見えました．実際，1980年代の初めにIBM PCとApple Lisaが登場し，PCが16ビット（今日では32ビット）へ移ることが明らかとなったのです．

しかしながら数百万台の8ビットマイコン，特にZ80は引続きプリンタ，ターミナル，データ・コミュニケーション，産業制御などの分野で使用されています．これらの分野の人達が望んでいるのは，16ビットではなくて下記の3つの目標を満たす新しい8ビット・プロセッサなのです．

- 高性能化と高集積化…性能面，コスト面のキーとなる項目を満たすこと，たとえば64Kバイトのリミットを破ること，部品点数が少なくなることなど．
- フル・コンパチビリティ…数百万ステップのソフトウェア投資が守れること，プログラマの訓練も不要なこと．
- CMOSプロセス…高速，低消費電力のシステム向けのプロセスとして．

現 在

この本を読まれるにつれ，皆様は64180が上記の目標を巧みに満たしていることに気付かれるでしょう．実際，64180は消え去るのではなく，長い，健全な寿命をもつ8ビットのマーケットを再認識させたのです．ここアメリカでは，64180はCBBS（Computer Bulletin Board System：コンピュータ化掲示板）からホーム・セキュリティ・システムまでのあらゆるシステムに，時にはレース・カー用のオンボードコンピュータとして，使われています．64180のメリットをフルに生かした新しいハードウェア，ソフトウェアが毎日のように出ており，熱心なユーザ・グループの支持を得ています．

64180は①高性能，②コスト・パフォーマンス，③低消費電力―これらは大多数の設計者が望むところですが―を要求される応用に理想的です．また，アーキテクチャが素直で理解しやすく，応用ソフトの開発も容易です．また，無償のものも含めて数百（あるいは数千？）の既存アプリケーション・ソフトウェアと言語もあります．

将 来

明るいというにつきましょう．64180は今後とも伸びる数多くの製品の中心として，長期にわたり使用され続けるでしょう．さて，この本により皆様は自分自身の応用に対し64180を適用するための勉強を開始できます．

Microfuture社 社長　トーマス W.カントレル

（編者和訳）

Recommendation

The Past

In the early '70s, the invention of the first 8-bit microprocessors changed the world forever. For the first time, the dream of a truly 'personal' computer decame possible.

The existing 'big' computer companies couldn't easily adapt to the concept of desktop computing. This left the market open for hundreds of tiny hardware and software companies, all driven by a vision of the future.

One of my earliest PCs was an IMSAI box containing an 8080A CPU board (running at 2MHz) and a 4Kbyte RAM board. 'Programming' was accomplished by entering the binary opcodes on the front panel switches. Of course, the resulting 'application program' was limited by the available 8 switches and 8 LEDs !

From these humble beginnings, it wasn't long before new technology was able to boost performance. The arrival of the enhanced Z80 processor, floppy disks, denser RAM and low-cost video terminais provided the 'platform' for the pioneering software products like CP/M (Digital Research), MBASIC (Microsoft), Wordstar (MicroPro) and dBASE (Ashton-Tate). These programs, all written in assembly language and hand tuned for speed and efficiency, have withstood the test of time. Even by todays standards, they stand as shining examples of the art of programming.

Inevitably, technology marched on and by the end of the decade, focus had switched to the battle for 16-bit dominance between the Intel 8088/8086, Motorola 68000 and Zilog Z8000. With resources devoted to new micros, it seemed the 8-bit market was being abandoned. Indeed,the early '80s introduction of the IBM PC and Apple Lisa confirmed that 'PCs' would move onward to 16-bits (and today, 32-bits).

Yet, millions of 8-bit micros, especially the Zilog Z80, were still being used in applications such as printers, terminals, data communications, industrial control, etc. What these customers needed was not 16-bits but rather a new 8-bit chip fulfilling three key goals.

- High Performance/Integration…The CPU had to break the key performance and cost barriers such as 64Kbyte memory limit and excessive chip count.
- Total Compatibility…To maintain the investment of millions of lines of code (and the resulting programmer training!).
- CMOS Process…Recognizing the '80s best process technology for fast, low-power systems.

The Present

As you read this book, you'll find that the Hitachi HD64180 has met these goals admirably-indeed, the '180 has 'revived' the 8-bit market which, rather than disappearing, has a long, healthy life ahead.

Here in the US, the '180 is being used in everything from a CBBS (Computer Bulletin Board System) to a digital video home security system-even an on-board instrumentation computer for a race car! New hardware and software which takes full advantage of the '180 emerges daily and is backed by an active network of user groups.

The '180 is ideal for applications requiring good performance, efficient design and low power-features most designers appreciate. Development is easy because the architecture is straight-forward and understood by many. Hundreds (even thousands) of different applications and languages are available-many for free!

The Future

…is bright, and the '180 will continue to be the centerpiece of an ever growing variety of products for many years. Now, with this book, you can start learning how to put the '180 to work in your own application.

Thomas W. Cantrell
President, Microfuture

はしがき

1971年，マイコンが最初に世の中に出てから十余年経過し，マイコンの応用は広い分野に定着し，またマイコンLSI自体も第3，第4世代へと着実な発展を見せています．マイコンLSIを仮に上位機種（16，32ビット），中位機種（8ビット），下位機種（シングル・チップ）に分けて流れを見ると，次のような動向が見られます．すなわち，上位機種では高性能化の追求が行われ，32ビット機種が登場してきており，中位機種では周辺機能を取り込んだシステム集積化による性能・コスト向上の追求が，また下位機種ではシステム集積化とともに各応用分野を指向した専用化が行われています．また，すべてに共通した事項として，低電力化を指向したプロセスのCMOS化が顕著な動向として挙げられます．上記のハードウェアの流れに加え，マイコンの世界で無視し得ないものとして，いわゆる標準OSとそのうえで走るソフトウェア財産の流れがあります．1978年ころ，8ビット・マイコン用として登場したオペレーティングシステムCP/Mはそのうえに数千点の応用ソフトウェアが蓄積され，標準OSとしての位置を占めました．その後，そのソフトウェア財産はCP/M-86，MS-DOSを通じ，16ビット・マイコンにも継承されています．CP/Mの走る8ビット・マイクロプロセッサで中心的な地位を占めているのがZilog社のZ80です．現在でも多くの8ビット・パソコンがZ80を使用しているのは皆様御存じのとおりです．64180はZ80を，CP/Mがより効率的に走ることも含め機能強化したシステム集積型の8ビット・マイクロプロセッサです．

64180の開発には以下に述べるようないきさつがあります．1982年の暮れ，米国Microfuture社のトム・カントレル（推薦のことば筆者，当時，日立アメリカ社）と筆者は，当時，CP/M-68Kの共同開発を行っていた米国ディジタルリサーチ社のソフトウェア技術者達と仕事の後に，雑談をしていました．その際，話題となったのが，8ビット・プロセッサのアドレス空間の制限－俗にいう64Kバイトの壁，さらに当時同社で開発中であったCP/M-Plus（V3.0）におけるバンク切替方式による本問題への対策，あるいはOS効率化のためのRAMディスク

などでした．その際，一同の間で期せずしてひらめいたのが，Z80を機能強化し上記問題を解決し，CP/M-Plusを効率的に走らせるプロセッサを開発してはという発想でした．本件はその後日立の技術者達に取り上げられ，具体化のための検討が行われ，さらに広いユーザの意見を聞いた結果できあがったのが64180のスペックなのです．その後の検討により数々の機能が付け加えられていますが，64180のMMUの方式，DMAによるメモリ↔メモリのブロック転送などには，上記の議論の結果を色濃く残しています．64180は，ソフトウェア技術者の提案を端緒としてこの世に出たプロセッサなのです．

この64180プロセッサですが，詳細は以下の章に譲るとして，大きくは次のような特徴をもっています．MMU，DMAを内蔵し，8ビット・プロセッサで問題のアドレス空間は1MBまで拡大されています．ネットワーク時代への備えとして，シリアル・インタフェース3チャネルをもっています．命令セットはZ80に対し，アップワードにコンパチブルで新たに乗算を含む12種の命令が付加されています．スリープ命令はCMOSプロセスとあいまって，低電力化の効果を一段と発揮するものです．何よりも，Z80コンパチブルの命令セットとバンク切替方式のMMUは，CP/M-Plus（V3.0）を効率的に走らせ，そのうえのソフトウェア財産を活用できる点で，64180の最大の特徴といえましょう．以上の点よりもわかるように，64180は8ビット機ではありますが色々な点で機能強化され，16ビットへの橋渡しをするマイクロプロセッサなのです．

本書は，主としてZ80を使用したことのある方々を対象としています．その意味でZ80と相違する点，付加した機能について特に注意して説明するよう心掛けました．前半に主としてハードウェア機能を，後半にはCP/M-Plus（V3.0）とのインタフェースを含む応用例について述べています．本書が皆様の64180の応用を進める際に良い手ほどきとなることを期待します．

昭和62年12月

編　者　し　る　す

目　　　次

第 I 編

C P U

64180は，Z80と上位コンパチブルな命令セットをもつCPUを中心に，DMAコントローラやMMUなどの周辺機能をワンチップに集積化したマイコンLSIです．第I編では，追加命令を中心としたCPUアーキテクチャおよびメモリインタフェースタイミングについて説明します．また，8ビットマイコンの弱点である，メモリ空間の制約（64KB以下）を解決するために導入されたバンク方式MMUの詳細について最後に説明します．いずれの項目も，64180を使ったシステムのソフト/ハード設計には必須の基本事項です．

1. 64180の概要

64180は，DMA コントローラや MMU などのシステムの構成に必要な周辺機能を内蔵した，8ビットでは世界で初めての「システム集積型」マイコンです．CPUは，Z80と上位互換性のある命令セットをもっていますが，ソフト/ハードの両面でZ80から改良されており，16ビットCPUにせまる性能を実現しています．各種の開発ツールもそろい，一方，製品のファミリー化も計画されており，8ビットの代表的CPUのひとつとして息の長い製品となるものと思われます．

1・1 システムオンチップ技術の流れ

〔1〕 **マイコン CPU の発展** 世界初のマイコン LSI4004 から10数年，この間半導体技術の進歩にささえられて，マイコン LSI は目覚ましい発展を遂げ，今や 32 ビットマイコンが実用化されるに至っています．32 ビットマイコンの華々しい開発競争に，ともすれば世間の注目が集まりがちですが，実際にはマイコンの出荷数量の 90 %以上を 8 ビット以下のマイコンが占めており，この分野でも目覚ましい進歩が続いています．32ビットマイコンを実現する半導体技術を，性能追求ではない別の目的に使ったらどうなるのか？それがシステムオンチップ化です．

〔2〕 **システムオンチップ化** 最初にシステムオンチップを実現したのは，4 ビットのシングルチップマイコンです．主に，ROM，RAM を内蔵してワンチップでコンピュータの機能を実現しました．シングルチップマイコンは，家電製品などに内蔵されてメカの制御などに使われています．たとえば，VTR などでは，マイコンの存在がなければ製品が成立しえないほど重要な部品となっています．一方，これらのシングルチップマイコンの流れに対し，1980 年代に入って少し異なった形のシステムオンチップ化を実現したマイコンが現れました．それが 80186です．このマイコンでは，メモリ以外でシステムの構成に必須の機能を集積化しています．いわば「システム集積型」マイコンといえます．

〔3〕 **システム集積型マイコン** なぜこのような「システム集積型」マイコンが出現したかは，マイコンの応用分野を考えてみるとわかります．**表 1・1** に示すように，マイコンの応用は大きく分けて，メカ制御中心の**コントローラ型**とデータ処理中心の**データ処理型**に二分できます．コントローラ型は，必要なメモリサイズが比較的小さくワンチップに集積化が可能です．すなわち，シングルチップマイコンとなるわけです．一方，データ処理型の応用の場合には，扱うデータ量が 64 KB 以上のことが多く，現在の技術ではワンチップ化が困難です．そこで，メモリ以外でシステム構成によく使われる，DMAC，タイマ，シリアルなどの標準的周辺機能を内蔵したマイコンが出現したわけです（**図 1・1** 参照）．64180 は，8 ビットでは世界初のシステム集積型マイコンです．

表 1・1　マイコンの応用分野の分類と使用されるマイコン

分類 / 項目	データ処理型	コントローラ型
特長	データ処理中心 コンピュータ・ライク	メカ制御中心 機器組込み型
応用機器の例	パソコン，ワークステーション タイプライタ，ワープロ	家電品(VTR, エアコンなど) 自動車エンジン制御 ロボット
必要なメモリ空間	大きい（64 KB 以上）	小さい（64 KB 以下）
内蔵機能への要求	メモリ以外のシステム構成に 必要な機能の内蔵	all-in-one
必要な内蔵機能	タイマ，シリアル，DMAC MMU，キャッシュメモリ リフレッシュコントローラ ウェイトコントローラなど	ROM，RAM タイマ，シリアル A/D 変換器 I/O ポートなど
使われるマイコンの流れ	マルチチップマイコン ⇩ システム集積型マイコン	シングルチップマイコン ⇩ ASIC マイコン

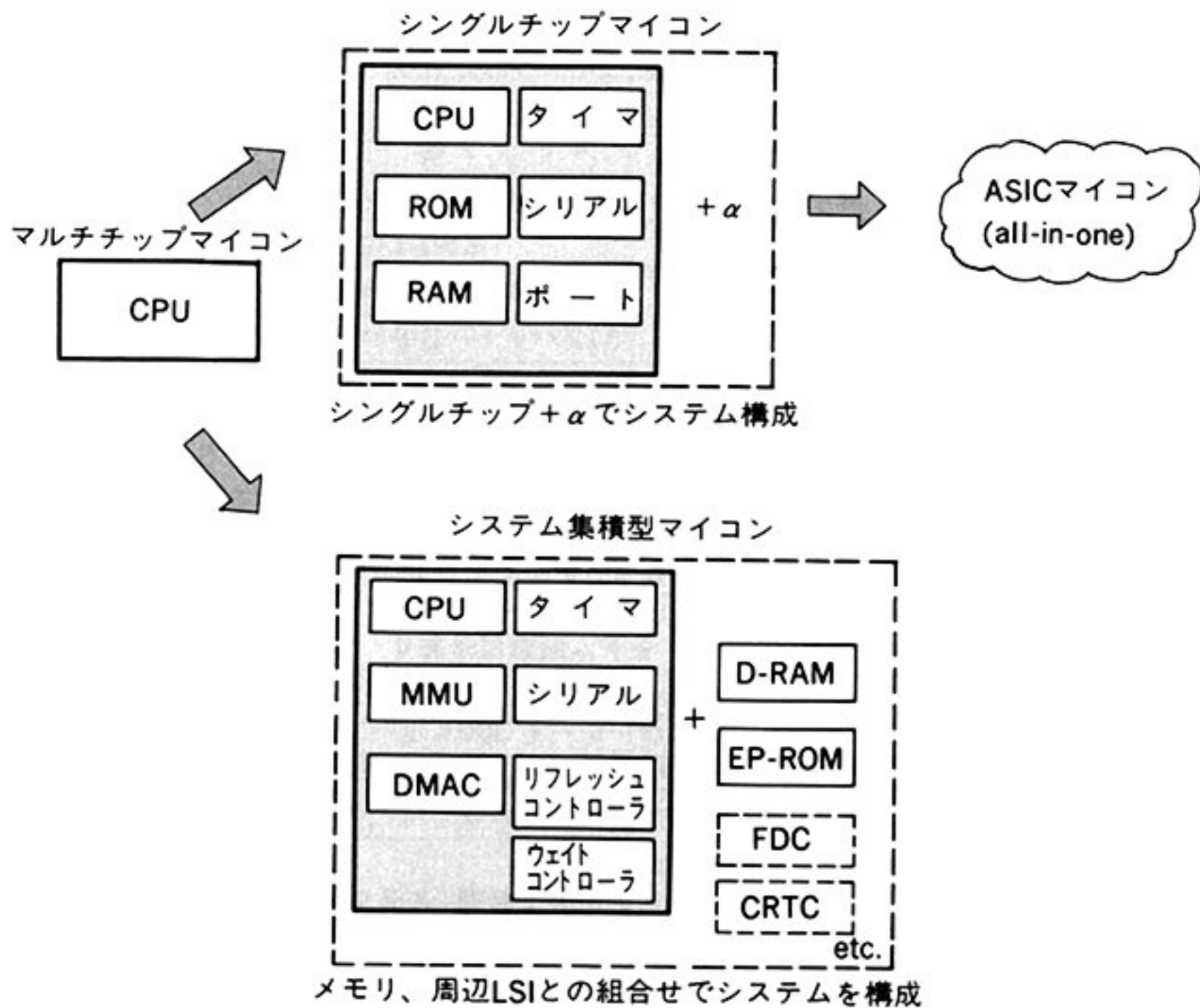



図 1・1　シングルチップマイコンとシステム集積型マイコン

1・2 64180の特長と内蔵機能

〔1〕 **なぜ8ビットシステム集積型マイコンか？**　データ処理型の応用では16ビットマイコンが主流になるのではないかと考えられる方も多いと思いますが，必ずしもそうはならない理由があります．まずLSIコストの問題があります．また，16ビット幅のバスはプリント基板上で場所をとりコスト高となります．まだまだメモリにおいて×16ビット出力のものは数少ないという問題もあります．また，周辺LSIは大部分が8ビットバスです．メモリ空間サイズを除けば，性能的には8ビットで十分という応用も数多くあります．また，8ビットでの多くのソフトウェア財産の存在も見逃せません．

表1・2　主要性能と機能

項目		仕様	
プロセス		CMOS	
特性	消費電力 (typ) (V_{cc}=5V f=4MHz)	通常動作時	50mW
		スリープ・モード時	30mW
		システム・ストップ時	12.5mW
	最小命令実行時間	300ns (f=10MHz)	
機能	命令体系	Z80上位互換	
	割込み	外部4本，内部8本	
	アドレス空間	1Mバイト（512バイト…DIPパッケージの場合）	
	DMAコントローラ	2チャネル ○メモリ⟷メモリ ○メモリ⟷I/O（メモリ・マップI/O） 最大転送レート 1.3 Mバイト/秒(f=8 MHz)	
	非同期シリアル通信インタフェース (ASCI)	2チャネル ○8種のデータ・フォーマット ○3種のエラー・フラグ ○モデム制御信号あり	
	クロック同期式シリアルI/Oポート (CSI/O)	クロック同期シフト・レジスタ1チャネル 最大ボーレート300Kボー(f=6MHz)	
	タイマ	16ビット・リロード・タイマ2チャネル タイマ出力あり（"0"，"1"，トグル出力）	
	バスウェイトコントローラ	外部端子による制御 内部レジスタによる制御（メモリ，I/O）独立に設定可）	
	リフレッシュコントローラ	プログラマブル非同期リフレッシュ 8ビットのリフレッシュ・アドレス(256K DRAM対応)	
	バスインタフェース	80系/68系(63系)とも可能	

〔2〕 **16ビットにせまる8ビットマイコン**　64180は，命令体系に8ビットの世界標準ともいえるZ80と上位互換性のある命令体系が採用されており，既存のソフトウェア財産を活用できます．また，8ビットマイコンの弱点であったアドレス空間の制限（64Kバイト以下）は，内蔵のバンク方式のMMUによって命令の互換性を保ちながら最大1Mバイトまで拡大されています．

乗算命令の追加，10MHzの高速動作とあいまって，まさに16ビットの性能にせまる8ビットマイコンといえます．

〔3〕 **内　蔵　機　能**　CPU，MMU以外の内蔵機能としては，**DMAコントローラ**，**シリアルインタフェース**，**タイマ**，**リフレッシュコントローラ**などがあります（**図1・2**，**表1・2** 参照）．特に，シリアルインタフェースが3チャネル内蔵されているところに大きな特長があります．これは，今後の応用機器を考えると分散処理の方向へ動くと考えられ，その場合各機能ごとにマイコンが入るこ

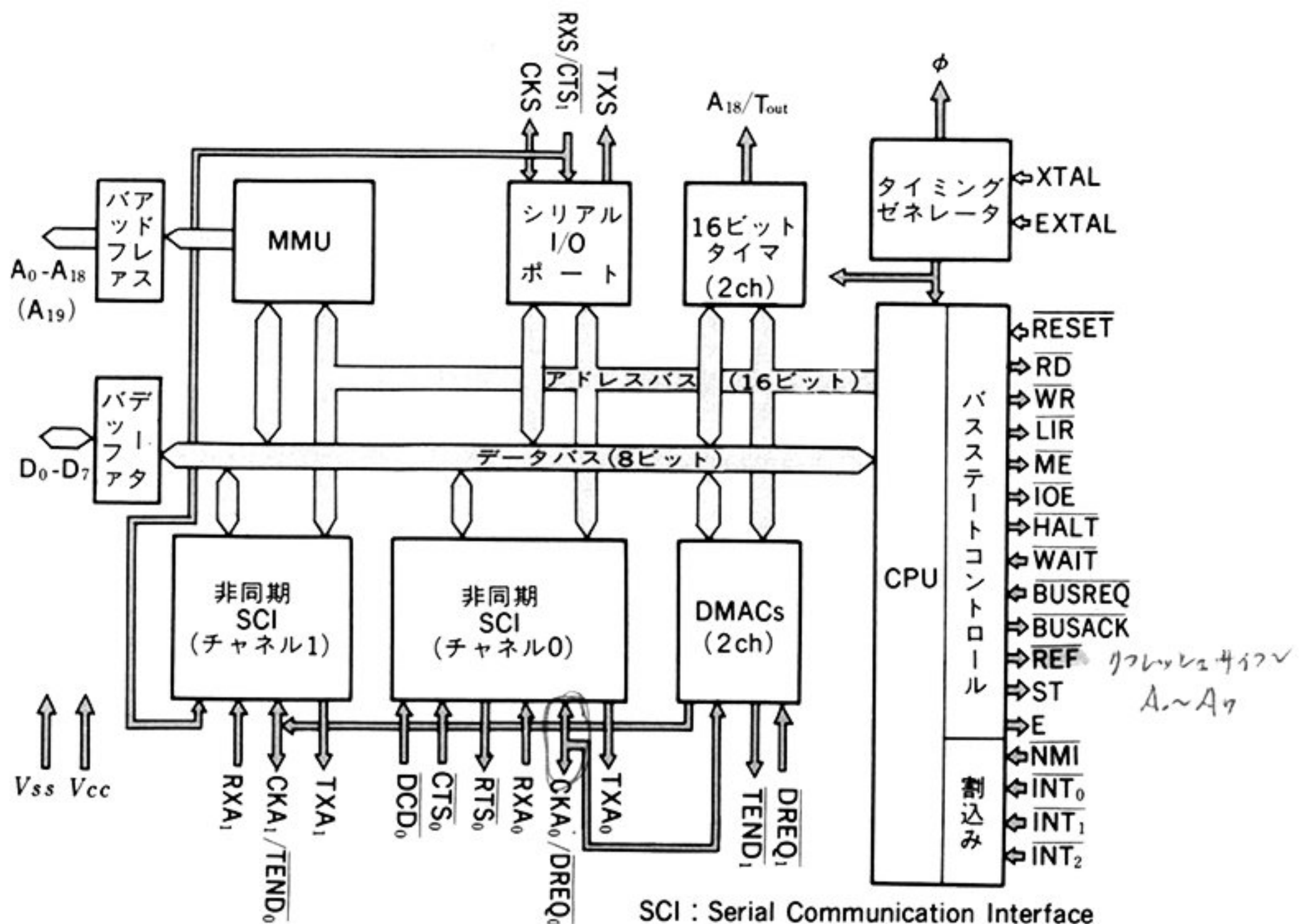



図1・2　ブロック図(内部アドレスは16ビット，MMUにより19ビット（20ビット）に拡張された物理アドレスが出力される)

とが予想されます．その際，各マイコン間のインタフェースはシリアルインタフェース経由で行われるのが普通です．上位 CPU との接続，サブ CPU との接続などを考えると，シリアルインタフェースは複数必要となります．

リフレッシュコントローラ，ウェイトコントローラの内蔵も大きな特長です．これらが内蔵されていることにより，ダイナミックメモリや擬似スタティックメモリとのインタフェースはごくわずかの外付 IC で済み，驚くほど簡単に実現できます．

1・3　ソフトウェアの継承性

〔1〕 **80系マイコン**　8ビットのマルチチップマイコンで最も広く使われているものが **Z80 CPU** です．国内では8ビットパソコンの大部分や MSX パソコンのメイン CPU に Z80 が使われているのは承知の事実です．この Z80 の命令セットは 8080 の命令を包含し，オブジェクトレベルで互換性を保ちながら機能拡張されています．64180 ではさらに，Z80 に上位互換性を保持しながら7種類12個の命令が追加されています．

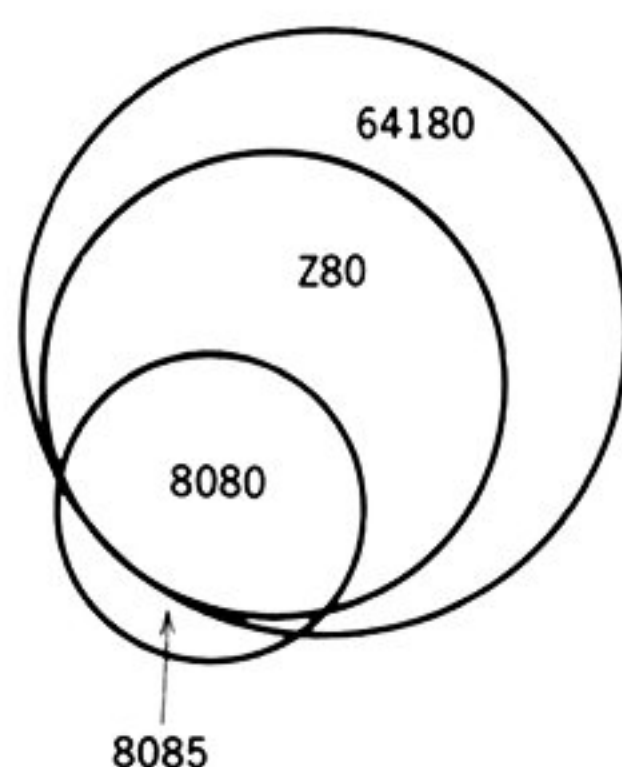
〔2〕 **64180 の追加命令**　64180 で強化された命令のうち，なんといっても効果が大きいものが**乗算命令**（MLT 命令）です．これにより8ビットの乗算は約1桁速くなりました．また，**スリープ命令**（SLP 命令）は CMOS の特長を生かした命令です．64180 は CMOS で作られているので，NMOS の標準的な Z80 に比べて消費電力はもともと約1/10（$f=4$ MHz時，50mW）ですが，この SLP 命令を使えば CPU はスタンバイ状態となり，さらに通常動作時の約1/4（$f=$

ニモニック	動　作	備　考
SLP	Sleep	
MLT ww	ww_R←wwHr×wwLr	ww＝BC，DE，HL
INOg,(m)	g←(m)	I/OアドレスのA_{15}～A_8は00H
OUTO(m),g	(m)←g	I/OアドレスのA_{15}～A_8は00H
OTIM	(C)←(HL),HL←HL＋1 C←C＋1,B←B－1	I/OアドレスのA_{15}～A_8は00H
OTIMR	(C)←(HL),HL←HL＋1 C←C＋1,B←B－1 Repeat until B＝0	I/OアドレスのA_{15}～A_8は00H
OTDM	(C)←(HL),HL←HL－1 C←C－1,B←B－1	I/OアドレスのA_{15}～A_8は00H
OTDMR	(C)←(HL),HL←HL－1 C←C－1,B←B－1 Repeat until B＝0	I/OアドレスのA_{15}～A_8は00H
TSTIO m	(C)∧m	I/OアドレスのA_{15}～A_8は00H
TST g	A∧g	gはレジスタ
TST m	A∧m	mは8ビットデータ
TST(HL)	A∧(HL)	

図1・3　80系マイコンの命令セットの関係と64180の追加命令一覧表

4 MHz 時，12.5 mW）まで下げることができます．スリープ命令はCPUを間欠動作させて平均消費電流を抑えたいような応用に便利です（**図 1・3**）．

［3］ **CP/M80 OS**　8ビットマイコンの代表的OSに，米国ディジタルリサーチ社の **CP/M80** があります．CP/M は 8080 系の 8ビット CPU を使うパソコンのほとんどの機種に採用されており，世界的標準となっています．パソコン以外でもワープロ，タイプライタなどCP/Mで動いているものが数多くあります（**図 1・4**）．

CP/Mには，一般的な CP/M2.2［Ver 2.2］と，機能改良された CP/M PLUS［Ver 3.0］があります．特にCP/M PLUSでは，バンクモードというメモリ空間の拡張機能がサポートされています．このバンクモードは，64180 のバンク方式 MMU を使えば外付回路なしで実現することができます．64180 は CP/M 2.2 のみならず，CP/M PLUS の搭載にも最適の CPU といえます．

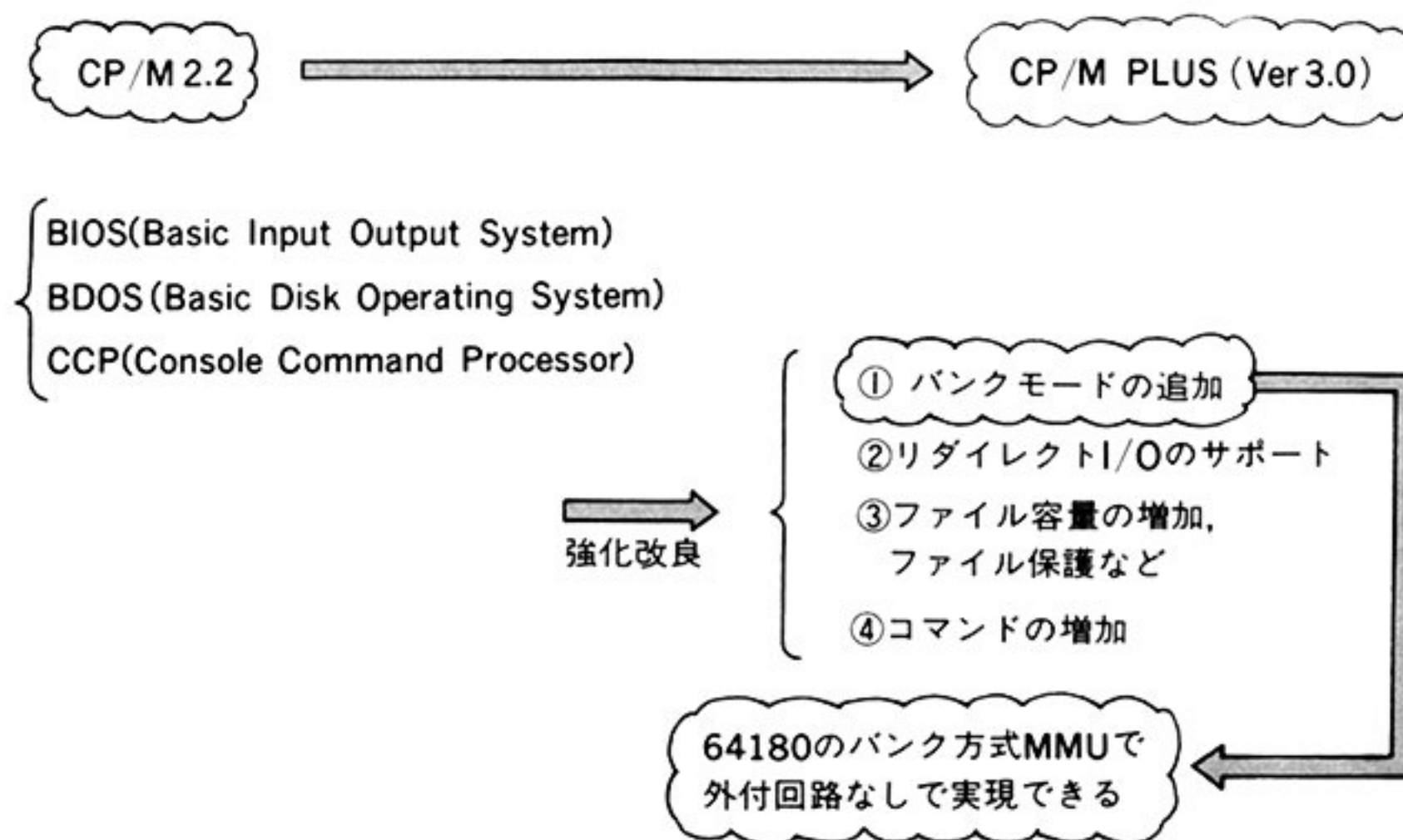



図 1・4　CP/M80 OS の発展

1・4 Z80との相違

64180には前節で述べたように，Z80に12個の命令が追加されています．また，各命令の実行サイクル数も約10〜20％短縮されており，6MHzバージョンでZ80の7〜8MHz版に匹敵する処理能力をもっています．

端子は，Z80に相当する端子が備わっていますが名前は一部異なっています．

$\overline{M_1} \longleftrightarrow \overline{LIR}$，$\overline{MREQ} \longleftrightarrow \overline{ME}$，$\overline{IOREQ} \longleftrightarrow \overline{IOE}$

バスタイミングは基本的にはZ80と同じですが，細部は異なります．特に，次に述べるリフレッシュ方式の違いにより，命令フェッチサイクルが3ステートとなっている点が一番大きな違いです．Z80ではこれが4ステートとなっており，第4ステートでリフレッシュを行っています．その他にも$\overline{ME}$, $\overline{IOE}$, $\overline{RD}$ のタイミングについて細かい違いがあります（付録参照）．

リフレッシュサイクルの挿入方法が改善されて，ダイナミックRAMとのインタフェースが格段にやりやすくなったことは，64180の大きな利点のひとつです．Z80の場合，M_1サイクルで必ずリフレッシュサイクルが入るため，特に高速版ではタイミング設計が非常に難しく，また，リフレッシュサイクルが不必要に頻繁に入ってしまうという問題がありました．64180では，リフレッシュサイクルは命令実行とは非同期に，一定サイクル間隔ごとに入る方式に改められています．リフレッシュ挿入の間隔や，リフレッシュサイクルの長さなどがプログラム指定でき，周波数が変わった場合も最適のリフレッシュ間隔を選ぶことができます．

割込み方式の改良も大きな利点のひとつです．Z80の場合，Z80系の周辺LSIを使う場合はデイジーチェイン方式が使えて簡単にシステムを組めますが，Z80系以外の周辺LSIの場合は設計が難しくなります．64180では，Z80と同一の割込み端子（$\overline{INT_0}$, $\overline{NMI}$）に加えて割込み端子が2本追加されており（$\overline{INT_1}$, $\overline{INT_2}$），これらに対応するベクタは内部で自動的に発生する方式をとっております．このため，Z80系以外の周辺LSIからの割込みの接続が非常に簡単にできます（**表1・3**）．

表 1・3 Z80との相違点

	Z80	64180
追加命令	—	6種類12命令追加 (SLP,MLT,TST,INO,OUTO,OTIM)
命令実行サイクル数	—	約10～20%サイクル数短縮 → 代表例
端子名	$\overline{M_1}$, $\overline{MREQ}$, $\overline{IOREQ}$	$\overline{LIR}$, $\overline{ME}$, $\overline{IOE}$
クロック発振器	外部クロック入力	発振器内蔵（水晶またはセラミックの振動子をつけるだけでよい）
バスタイミング	—	$\overline{ME}$, $\overline{IOE}$, $\overline{RD}$の出力タイミングが異なる
リフレッシュサイクルの挿入方法	各命令フェッチサイクルの第4サイクルに必ず挿入（命令同期式）	一定サイクル間隔ごとに命令実行に非同期に挿入（命令非同期式）リフレッシュインターバルを周波数によらず一定にできるので不必要に頻繁なリフレッシュを避けることが可能．リフレッシュサイクルの長さも2または3ステートが選べ，リフレッシュサイクルを全く挿入しないモードも可能
割込み	2本：$\overline{NMI}$, $\overline{INT}$ デイジーチェイン方式	4本：$\overline{NMI}$, $\overline{INT_0}$, $\overline{INT_1}$, $\overline{INT_2}$ デイジーチェイン方式も可能（$\overline{INT_0}$）だが，$\overline{INT_1}$, $\overline{INT_2}$および内蔵I/Oからの割込みはベクタ番号を内部で発生． 固定ベクタ，固定優先順位の割込みコントローラを内蔵
アドレス空間	64KB	512KB（面実装パッケージでは1MB） 内蔵のバンク方式MMUによる論理空間（64KB）から物理空間への写像により最大1MBを直接アクセス可能
低消費電力モード	—	スリープモード/システムストップモード
ウェイトステート制御	$\overline{WAIT}$端子のみ	$\overline{WAIT}$端子の他にプログラマブルウェイト可能
内蔵I/O機能	—	DMAC，タイマ，シリアル

代表例

命令	Z80	64180
ADD A,(HL)	7	6
ADD IX,xx	15	10
BIT b,(HL)	12	9
LD A,(BC)	7	6
LD(BC),A	7	7
LD ww,mn	20	9
JR$_j$	12	8

1・5 応用分野

以上述べてきたような優れた特長をもつ64180の応用としては，次のような分野が考えられます（**表1・4**）．

(1) 従来から80系のマイコンを使用してきており各種のソフトウェア財産が存在するが，アドレス空間が64Kバイトでは不足してきており，**16ビットを使うべきかどうか迷っている分野**．ハンディー日本語ワープロ，プログラマブルシーケンサ，FAXといった機器がこれにあたります．特に，ダイナミックRAMや擬似スタティックRAMを使うシステムや，All-CMOS化して低消費電力動作を狙う分野に適しています．

(2) 従来から80系のマイコンを使用しており，アドレスは64Kバイト以下でよいが，タイマ，シリアルなどの周辺LSIを多用している分野．**機器のサイズの関係で基板をできる限り小さくしたいし，消費電力ももちろん下げたい**．このような分野では，64180の内蔵機能が基板サイズ縮小に大きな効果があります．

(3) 従来8ビットシングルチップマイコンを使っていたが，制御が複雑となり，**メモリ容量が増加しシングルチップに入りきらなくなってきている分野**．しかし，できる限りシングルチップマイコンと同じように簡単な回路構成としたい．たとえば，電子タイプライタ，電子はかり，券売機，PPCなどがこれにあたります．

(4) 応用機器の性質として，**シリアルインタフェースが少なくとも2本以上必要な応用**．たとえば，端末，POS，モデム制御，自動車電話，キーテレホンなどがこれにあたります．

表1・4　応用分野と応用機器の例

分　野	応用機器
民　生	電子楽器，無線機
産　業	プログラマブルシーケンサ，電子はかり，自動販売機，券売機， 自動車パネル表示器
OA，情報	MSXパソコン，プリンタ，電子タイプライタ，ハンディー日本語ワープロ， 端末，POS，ECR，XYプロッタ，PPC
通　信	FAX，キーテレホン，PBX，モデム制御，パソコン通信，自動車電話

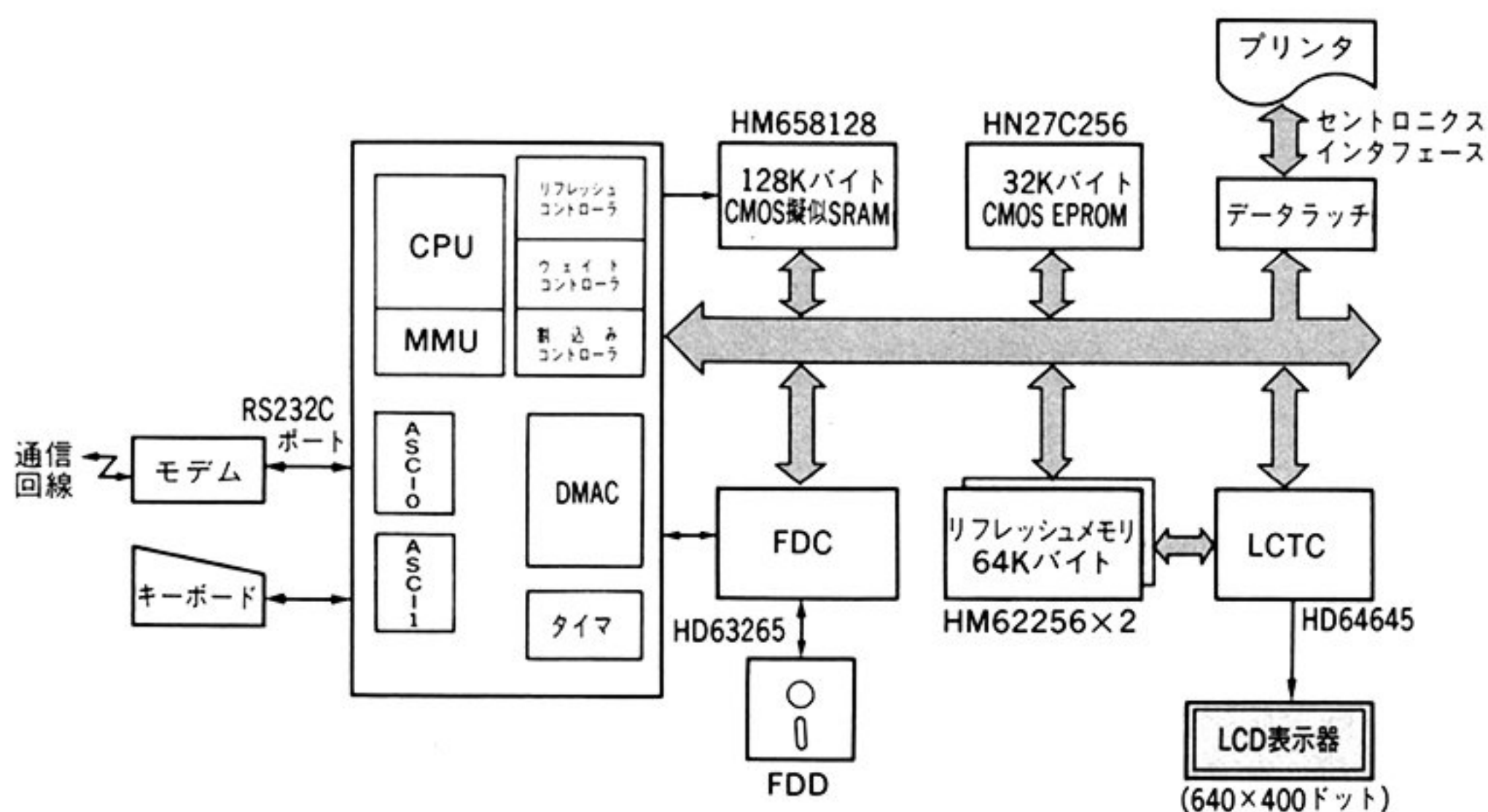



図 1・5 液晶表示つきワープロ/パソコンへの応用例

以上のような例の他にも，一般に高速処理，大きなアドレス空間，タイマ，シリアルなどの周辺機能が必要な応用など，幅広い応用が考えられます．Z80と同様に8ビットマイコンの標準品として，息の長い製品となるものと考えられます（**図1・5**）．

1・6 開発環境

どんなに優れたマイコンLSIであっても，開発装置がなければシステムを開発することはできません．その点64180は，半導体メーカである日立製作所以外にも，数多くの開発ツールメーカからサポートツールが発売されています．廉価普及版から高性能高機能型まで用途に応じて選ぶことができます(**表1・5**)．

開発ツールの形態は大きく分けて，専用の開発装置をホストとするタイプと汎用のパソコンやミニコンをホストとするタイプに分けられます．専用型の代表例はYHP社の64000などがそれです．専用型はホストが開発装置専用なのでコスト的に割高となりますが，一般に，機能，性能，使い勝手に優れています．

一方，汎用ホスト型は日立製作所のASEなどがそれにあたります．汎用ホスト上でエディト，クロスアセンブルを行い，RS232Cのシリアルインタフェース経由でロードモジュールを専用のリアルタイムエミュレータ上にロードしデバッグを行います．汎用型はコスト的には安くなりますが，大きなプログラム開発になるとRS232Cによるダウンロード時間がネックとなる場合があります．

最近では汎用型でも，ホストをたとえばPC9801に固定しパソコン内に専用基板を差し込んで，エミュレータとのインタフェースをパラレルバスで行い，ダウンロード時間のネックを解消した製品もあり（岩崎技研工業社製のPROICEなど）ますが，パラレルバス方式は，ホストが限定されており専用型と汎用型の中間といえます．

また，各社ともユーザの負担をできる限り下げるために，システムの一部分のボードなどを取り替えることによって各種のマイコンに対応できるようにしてあるものが多くなっています．

予算の関係で64180用の開発ツールを購入できない場合には，CP/Mののっているパソコンと Z80 用のアセンブラでも十分開発は行えます．たとえば，Microsoft社のMACRO80のようにマクロ機能つきのアセンブラであれば，64180の拡張命令をマクロ定義して使うこともできます．また，CP/Mに内蔵のデバッガでも十分プログラムデバッグはできます．不足分はロジックアナライザで補えばよいでしょう(**図1・6**)．

表 1・5　64180開発ツール一覧表

メーカ名	ホストコンピュータ	リアルタイムエミュレータ	クロスソフトウェア		
			アセンブラ	シミュレータ	C コンパイラ
日立製作所	VAX-11 SD200 IBM-PC	○	○	—	○
横河ヒューレットパッカード (YHP)	HP64000	○	○	—	—
岩崎技研工業	PC9801 IBM-PC	○	○	—	○
ソフィアシステムズ	SA3000 ES1000	○ (SA600)	○	—	—
横河電機 (YEW)	HP64000 VAX-11, MDS パソコン	○ (3503)	※	※	※
ZAX コーポレーション	THE BOX-ER PC9801, IBM-PC	○	○	—	○
ユニダックス	PC9801	○	※	※	※
日本データゼネラル	ECLIPSE MV ファミリー	(○) 他社製品接続	○	○	○
アドテックシステムサイエンス	—	○	—	—	—
MICROTEC (日本マイクロテックリサーチ)	VAX-11 PC9801 IBM-PC, 他	—	○	○	○
コアデジタル	PC9801 IBM-PC, IBM5550 B16, HP64000 VAX-11	○	※	※	※
アンリツ	VAX-11, PC9801 IBM-PC HP64000 CP/M	○	※	※	※
ソニー・テクトロニクス	ロジックアナライザ 1240/1241型 +プローブ	—	—	—	—

※　マイクロテック・リサーチ社製他，市販ソフトを使用

(注)　上表の他に，下記各社製の64180用ソフトウェアが販売されています(( )内：販売会社)
エイ・ディ・シー(ケミコン・セミコンダクターズ・セールス)，AVOCET(工人舎)，ソフトウェアアシスト(住友商事)，ソフトマート，BSO(エーエスアールインターナショナル)，2500AD (アドテックシステムサイエンス)，IAR(ライフ・ボード)，ファームウェアシステム

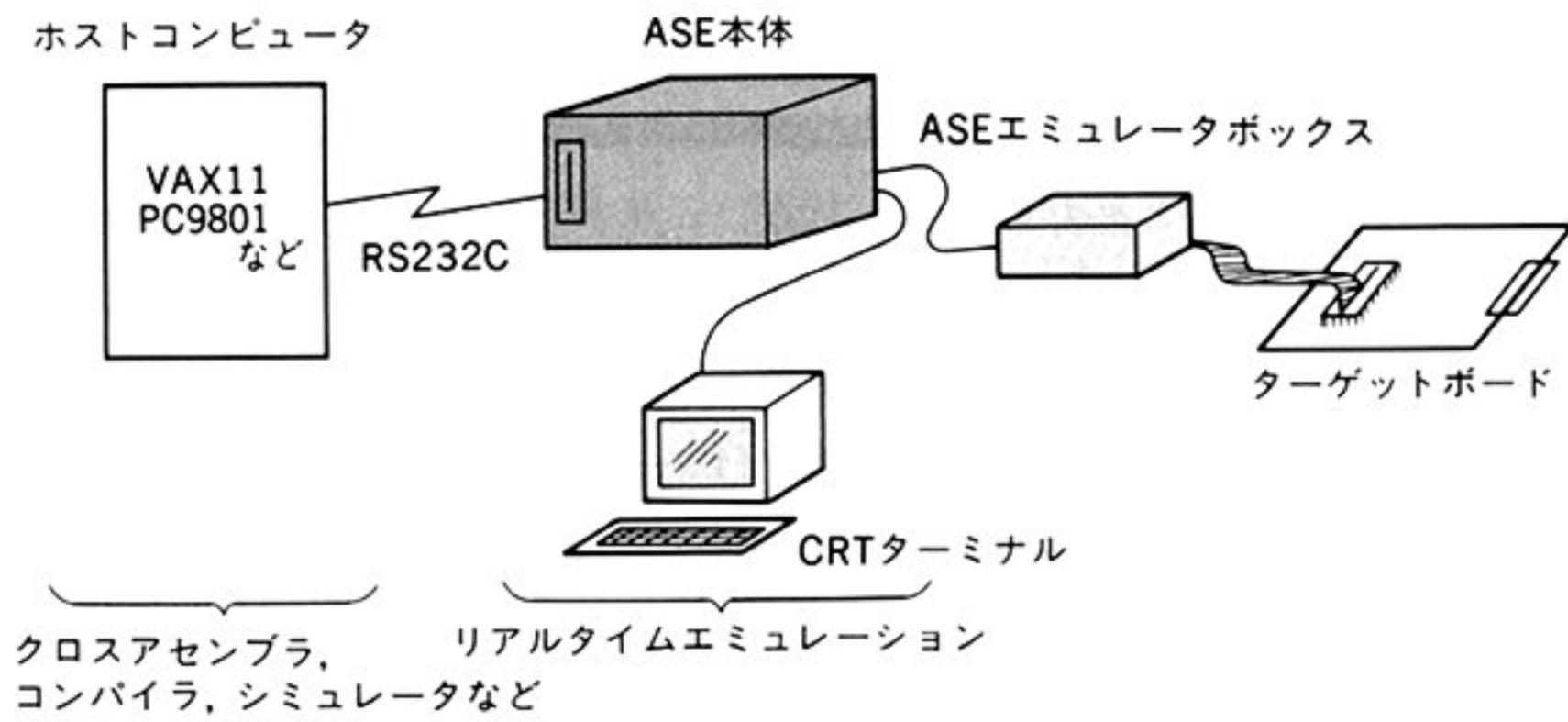



図 1・6　64180 開発環境の例

1・7 ファミリー展開

64180が発表されて約3年がたちましたが，この間，スピードアップ(10 MHz)，メモリ空間のアップ（1Mバイト）をはかり，初期のバージョンの不具合点の改良を行った“R1”バージョン，また，Z80 系の周辺 LSI とのインタフェースを改良した“Z”バージョンなど改良の努力が続けられています．64180 は，Z80 の本家の Zilog 社からもセカンドソース供給されています．“Z”バージョンは Zilog 社では“Z180”と呼ばれています．

1987 年春には注目すべき新製品が発表されました．64180 の全機能はそのままで 16 K バイトの EPROM，512 バイトの RAM，タイマ，コンパレータなどをオンチップ化した 647180X がそれです．647180X は日立製作所が提唱した **ZTAT** (Zero Turn Around Time) マイコンシリーズのひとつであり，いわゆる OTP (One Time Programable) マイコンです．ZTAT マイコンは，紫外線消去の EPROM 内蔵マイコンをプラスチックパッケージに封止したものです．1 回しかプログラムできないかわりに，従来のマスク ROM 型のシングルチップマイコンに近い価格で手に入れることができます．ZTAT マイコンは，少量多品種生産やプログラムバグが収束するまでの量産立上り時期に使うと効果的です．647180X は，64180 ファミリーの応用範囲を大きく広げる製品といえます．

64180 系としては，このほかにも ZTAT シリーズの品種追加，通信機能の強化版などの標準品展開，および ASIC 展開（カスタム対応）も計画されているようです．

1987 年春には以前から仕様が公表されていた Z800 が，Z280 という名称で Zilog 社より製品発表されました．Z280 は Z80 の CPU を 16 ビット化したものであり，Z80 とオブジェクトレベルのコンパチビリティがあります．キャッシュメモリを内蔵しており，現行市販されている 16 ビットよりもかなりの高性能を実現しています (図 1・7)．

64180 や Z280 などの開発により，Z80 系マイコンは 8 ビット標準マイコンの主流としての地位をここ当分保ち続けるものと思われます．

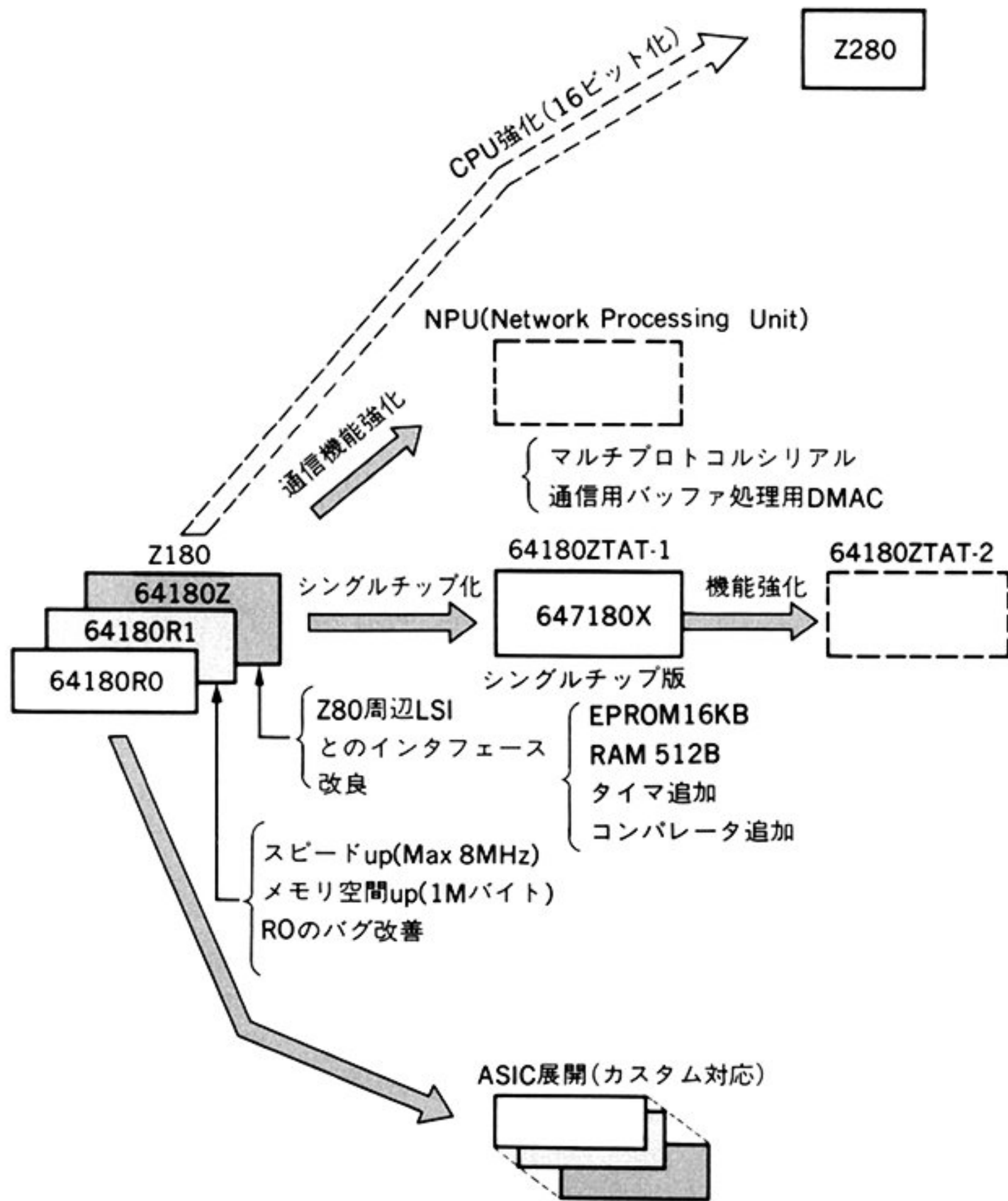



図 1・7 64180ファミリーの展開

2. CPU アーキテクチャ

64180は，Z80に対し上位コンパチブルなCPUをもっています．全体的に，命令の実行ステート数は短くなっています．また，乗算命令やスリープ命令（低消費電力モードに設定）などが新規に追加されています．さらに，DRAMのリフレッシュはオペコードフェッチとは独立しています．このように，64180のCPUはZ80に対しパフォーマンスが高くなっています．

2・1 レジスタ構成

〔1〕 **概　　要**　64180のCPUレジスタは，汎用レジスタセットと専用レジスタセットがあります．64180のプログラミングでは，これらのレジスタをどのように使いこなすかが重要になります（**図2・1**）．

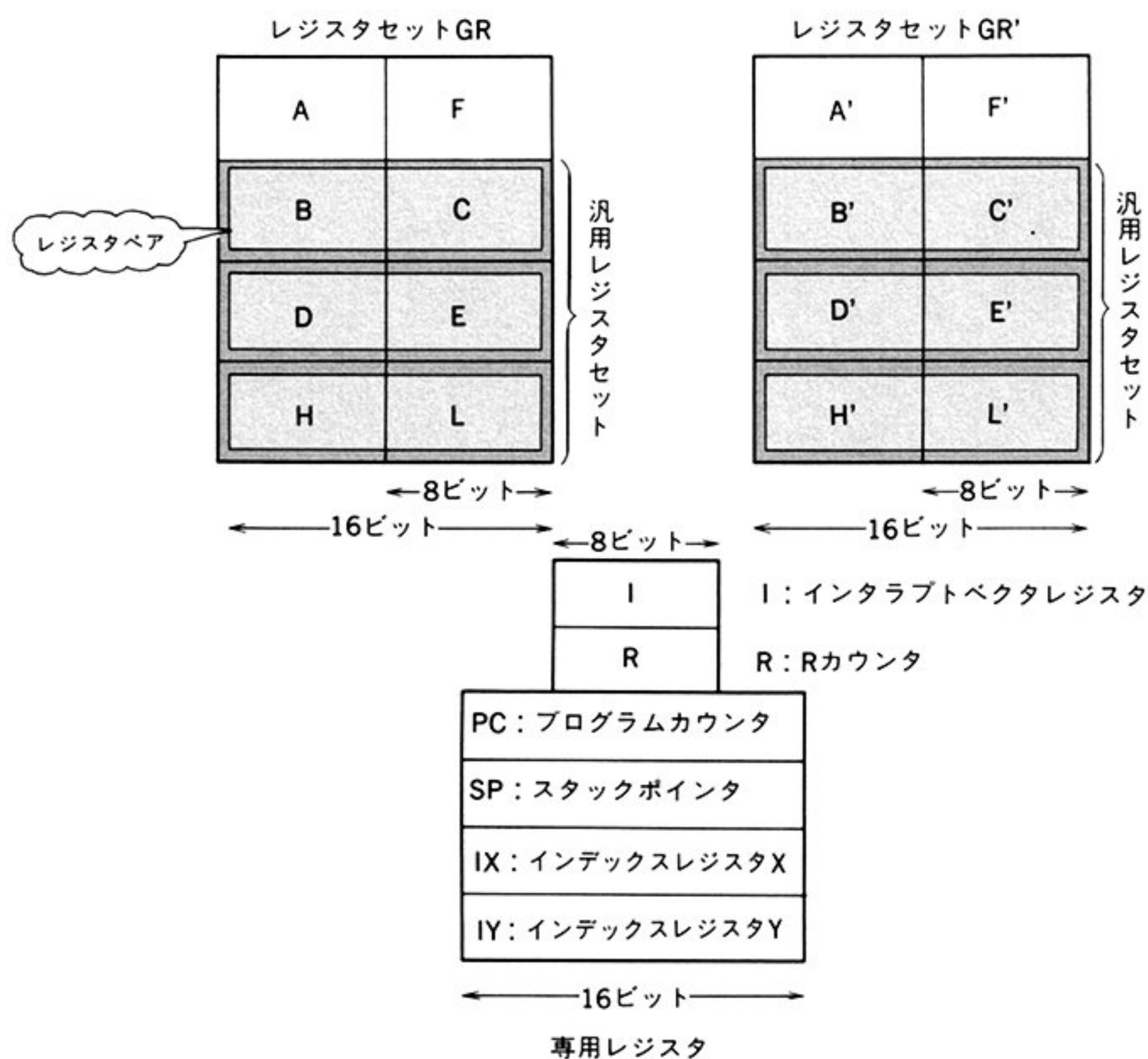



図2・1　レジスタ構成

〔2〕 **汎用レジスタ**　全く同一機能のレジスタセットが2組あります．汎用レジスタは8ビット長で，BとCレジスタ，DとEレジスタ，HとLレジスタにて，それぞれ16ビットのレジスタペアとしても使用できます．レジスタペアは，メモリやI/Oなどのアドレシングレジスタとして使う場合が多いようです．また，アキュムレータAは算術論理演算などに使用する重要なレジスタです．フラグレジスタFの各ビットの値は，算術論理演算後のアキュムレータの内容に従って設定されます．汎用レジスタは，対応するレジスタ間でその内容を交換することができます．この操作は専用命令（EXX；EX AF, AF'）を使います．割込みやサブルーチンコール後のレジスタ退避に使用して効果があります．

〔3〕 **専用レジスタ**　用途はあらかじめ決められています．PCは，次に実行される命令のアドレスを格納するプログラムカウンタです．SPは，割込みやサブルーチンコール後の復帰アドレスを格納する，スタック領域の先頭アドレスを示すスタックポインタです．さらに，IXとIYはインデックスアドレシングや16ビット演算に使用するインデックスレジスタで，ともに同一機能です．**Iレジスタ**は，割込みにおけるベクタ（16ビット）の上位8ビットを格納するインタラプトベクタレジスタです．**Rカウンタ**はオペコードフェッチサイクルごとにインクリメントするカウンタです．Rカウンタは Z80の場合とは異なり，リフレッシュアドレスとは無関係です．

PC，SP，IX，IYは16ビット長のレジスタで，IとRは8ビット長のレジスタです．

2・2 フラグレジスタ

〔1〕 **概　　要**　フラグレジスタFは6つのフラグで構成されており，8ビットや16ビットの算術論理演算を実行した結果に従って値が設定されます．これらのフラグは，命令により「変化する」「変化しない」「不定」のどれかをとります．未定義の2ビットは常に不定です．

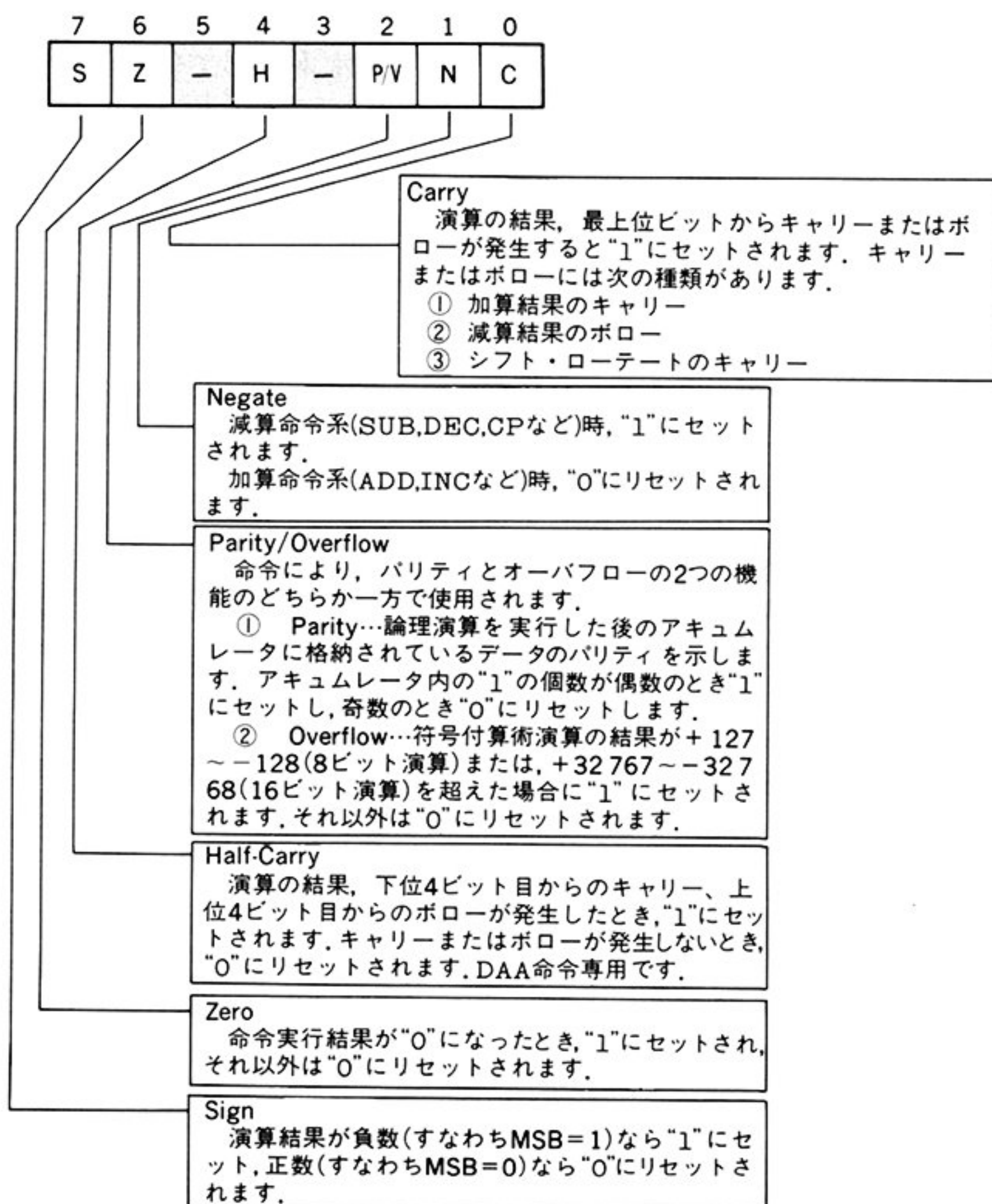



図2・2 フラグレジスタ

〔2〕 **キャリー：C**　最上位ビットからのキャリーまたはボローにより"1"にセットされます．図2・2のようなケースでキャリーやボローが発生します．キャリーについては他のレジスタに影響を与えずにキャリーフラグを変化させる特殊な命令（CCF，SCF）があります．

〔3〕 **ネゲート：N**　減算命令系の実行時に"1"にセットされ，加算命令系の実行時に"0"にリセットされます．

〔4〕 **パリティ・オーバフロー：P/V**　命令ごとにパリティまたはオーバフローのどちらか一方の機能として働きます．パリティはアキュムレータ内の"1"の数が偶数のときに"1"にセット，奇数のときに"0"にリセットされます．オーバフローは，符号付算術演算の結果が定義されている範囲を超えたとき"1"にセットされます．

〔5〕 **ハーフキャリー：H**　下位4ビット目からのキャリーや，上位4ビット目からのボローが発生したとき，"1"にセットされます．10進演算の補正命令（DAA命令）の実行時に使用します．

〔6〕 **ゼロ：Z**　命令の実行結果が0になったときに"1"にセットされます．また，ビットテスト命令でも影響を受けます．

〔7〕 **サイン：S**　演算結果が負の数ならば"1"にセットされます．

2・3 命令の構成

〔1〕 概　　要　64180の命令体系はZ80上位コンパチブルであり，命令は1バイトから4バイトで構成されています．オペコードは，2バイトまたは3バイトに渡る命令をもつ点が特徴です．この場合，オペコードはEDH，CBH，DDH，FDHのいずれかが先頭にきます．

表2・1　オペランドを伴わない命令の構成

バイト数	命令の構成	代表的な命令	
1	オペコード	LD g, g′	01 | g | g′
2	第1オペコード 第2オペコード	SLP	EDH 76H

g，g′：汎用レジスタ

表2・2　オペランドを1バイト伴う命令の構成

バイト数	命令の構成	代表的な命令	
2	オペコード オペランド	LD g, m	00 | g | 110 m
3	第1オペコード 第2オペコード オペランド	OUTO(m), g	EDH 00 | g | 001 m
		LD g, (IX+d)	DDH 01 | g | 110 d
4	第1オペコード 第2オペコード オペランド 第3オペコード	RLC(IX+d)	DDH CBH d 06H

g：汎用レジスタ，m, n：イミーディエイトデータ，d：ディスプレースメント

表 2・3 オペランドを2バイト伴う命令の構成

バイト数	命令の構成	代表的な命令	
3	オペコード / 第1オペランド / 第2オペランド	LD(mn), A	32H / n / m
4	第1オペコード / 第2オペコード / 第1オペランド / 第2オペランド	LD(IX+d), m	DDH / 36H / d / m
		LD(mn), IX	DDH / 22H / n / m

m, n：イミーディエイトデータ，d：ディスプレースメント

〔2〕 **オペランドが伴わない命令の構成**　1バイトまたは2バイトの命令です．2バイト命令では，第1オペコードと第2オペコードがあります．たとえば**表2・1**のように，SLP命令ではEDHが第1オペコードで，76Hが第2オペコードです．また，LD g, g′ 命令のように，命令コードの中にレジスタを識別するための3ビットのフィールドをもっているものもあります．

〔3〕 **オペランドを伴う命令の構成**　オペランドは1バイト（**表2・2**）または2バイト（**表2・3**）の長さであり，イミーディエイトデータ m, n やディスプレースメント d となります．オペランドが2バイトの場合には2バイトのイミーディエイトデータ mn または，ディスプレースメントとイミーディエイトデータで構成されています．オペランドが2バイト伴う3バイト命令は，mnで構成され d はありません．

なお，オペランドが伴う命令でもLD g, mのように，オペコードの中にレジスタを識別するフィールドをもつ命令があります．

〔4〕 **インデックスレジスタを使った命令**　**図2・3**に示すように，HL または（HL）をオペランドとしてもつ命令のみ，HL→IXまたはIY，（HL）→（IX+

図 2・3　インデックスを使った命令の構成

d）または（IY+d）と置き換えると，同じオペレーションでインデックスレジスタを使った命令となります．オペコードはIXの場合先頭にDDHを，IYの場合ではFDHを付加します．ただし例外もあり，JP(HL)の場合はHL→IXまたはIYに置き換えます．また，「EX DE, HL」の場合は置き換えると未定義コードとなります．

〔5〕 **未定義コード**　第1オペコードには未定義コードがありませんが，第2オペコードまたは第3オペコードには未定義コードがあります．64180では，未定義コードをCPUがフェッチすると，TRAP割込みが発生します．詳細は，「9·7 ノンマスカブルな割込み」を参照してください．

2・4 データの配置

64180の命令では，転送命令やプログラム制御命令などで指定されるメモリやレジスタへの8ビットデータの配置には，次のような場合があります．

〔1〕 **レジスタとイミーディエイトデータ**　図2・4のように，オペコードの次に配置されたイミーディエイトデータnが，レジスタペアや16ビットレジスタの下位8ビットに対応します．上位8ビットはmが対応します．

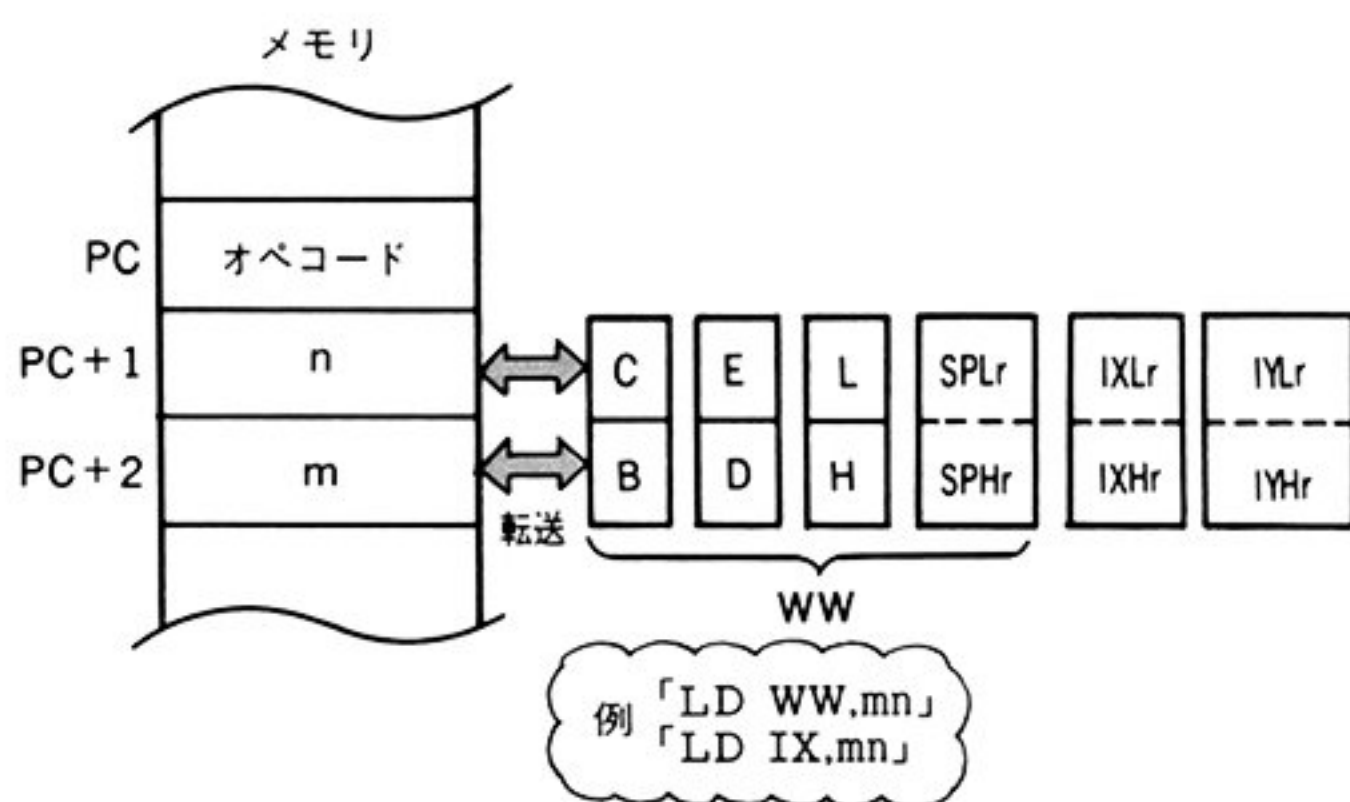



図 2・4　レジスタとイミーディエイトデータmnとの対応関係

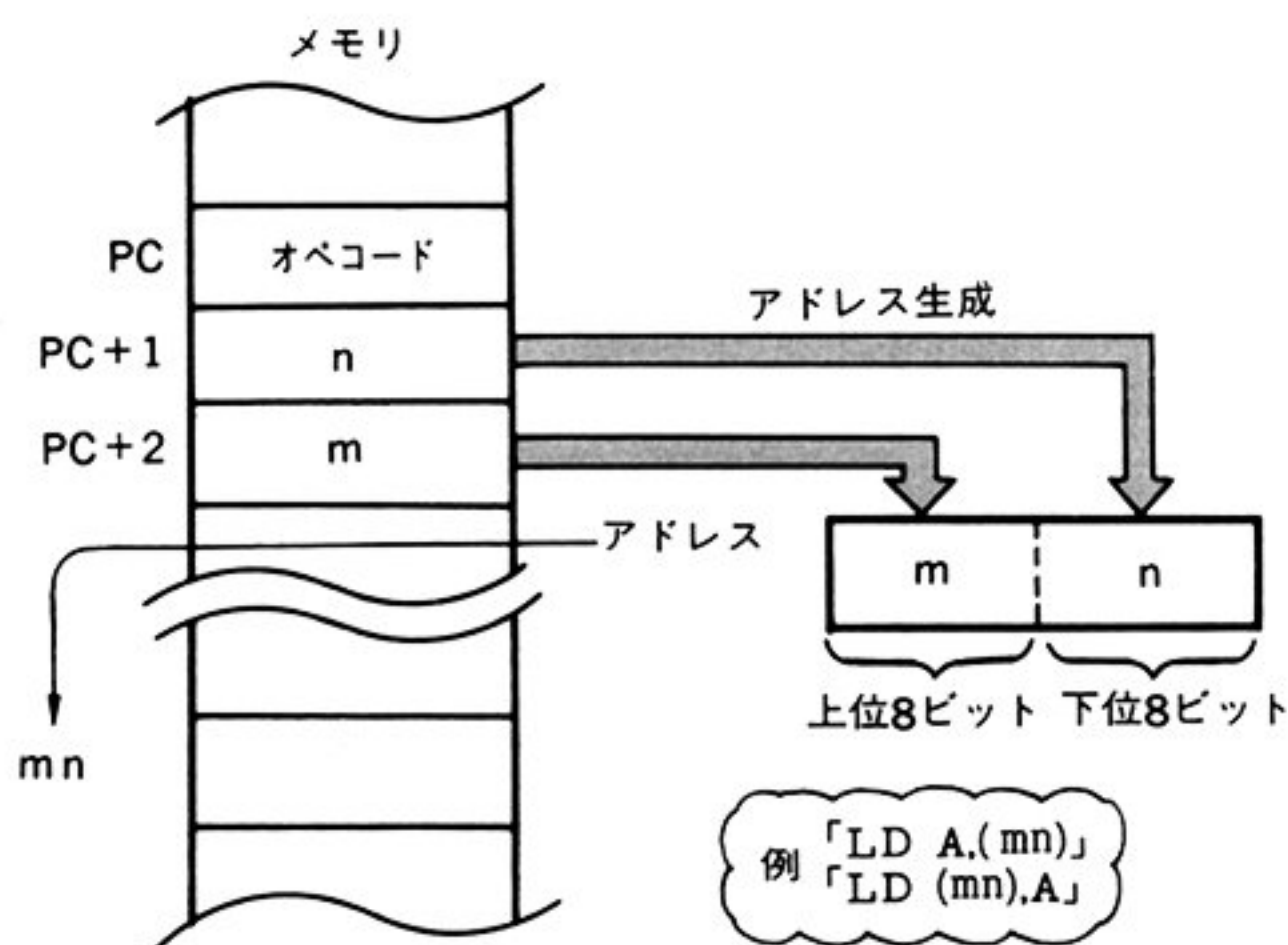



図 2・5　イミーディエイトデータによるアドレスの生成

〔2〕 **アドレス生成**　イミーディエイトデータmnによるアドレス生成です．図2•5のように，アドレスの上位8ビットはmによって，下位8ビットはnによって生成されます．

〔3〕 **レジスタと間接アドレシングによるメモリデータとの対応**　LD IX,(mn) 命令時です．アドレス生成は図2•5と同じであり，そのアドレスで示されるメモリとレジスタとの対応は図2•4と同等です．ただし，この時PCはmnに置き換えます．

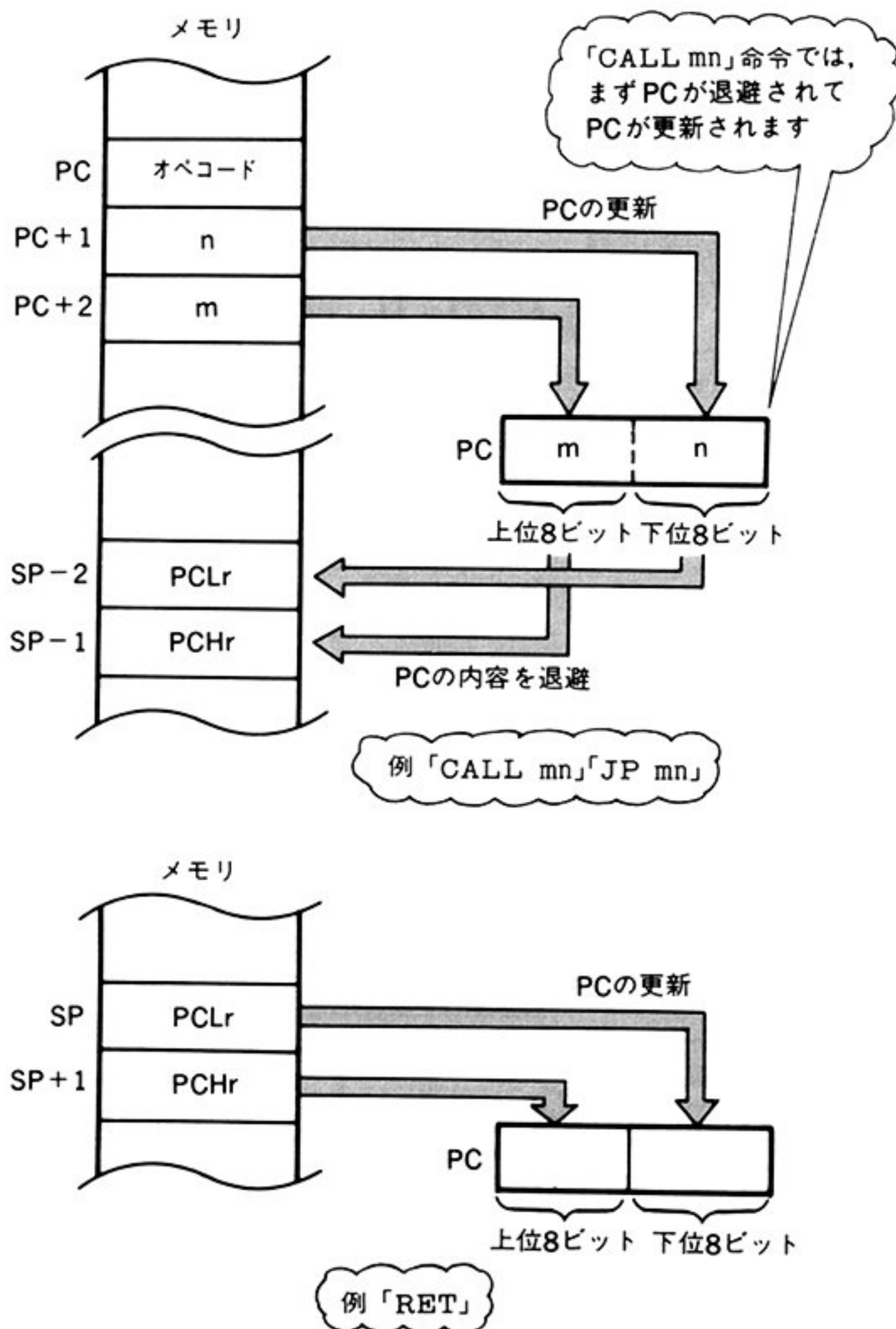



図2・6　プログラムカウンタPCの更新

〔4〕 **プログラムカウンタPCの更新**　図2・6に示します．PCの下位8ビットには，「CALL mn」「JP mn」命令の場合はイミーディエイトデータnが，「RET」命令の場合はスタックポインタSPで示されるスタックの内容が転送されます．上位8ビットにはそれぞれの命令で，イミーディエイトデータmまたはSP＋1で示されるスタックの内容が転送されます．「CALL mn」命令では，その前にPCの内容がスタックへ退避されます．

〔5〕 **レジスタとスタックとの対応**　図2・7に示します．レジスタペアや16ビットレジスタの上位8ビットは，PUSH命令ではSP－1で示されるスタックの内容が，POP命令や交換命令ではSP＋1で示されてスタックの内容が対応します．下位8ビットは，それぞれSP－2とSPで示されるスタックの内容が対応します．

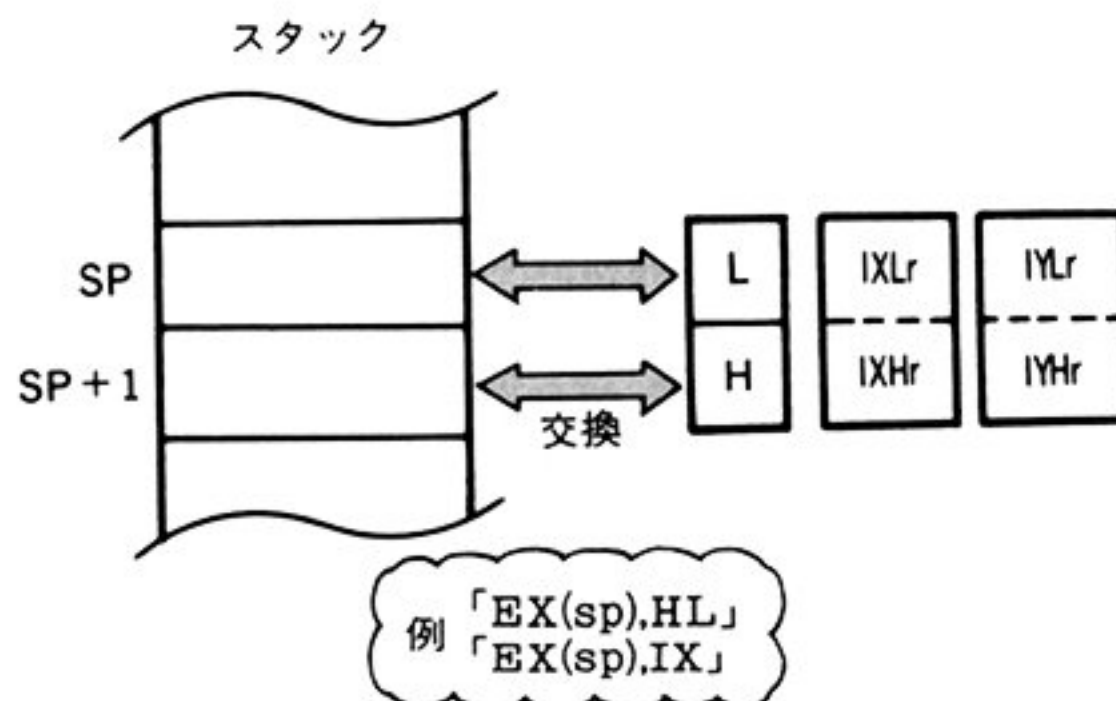
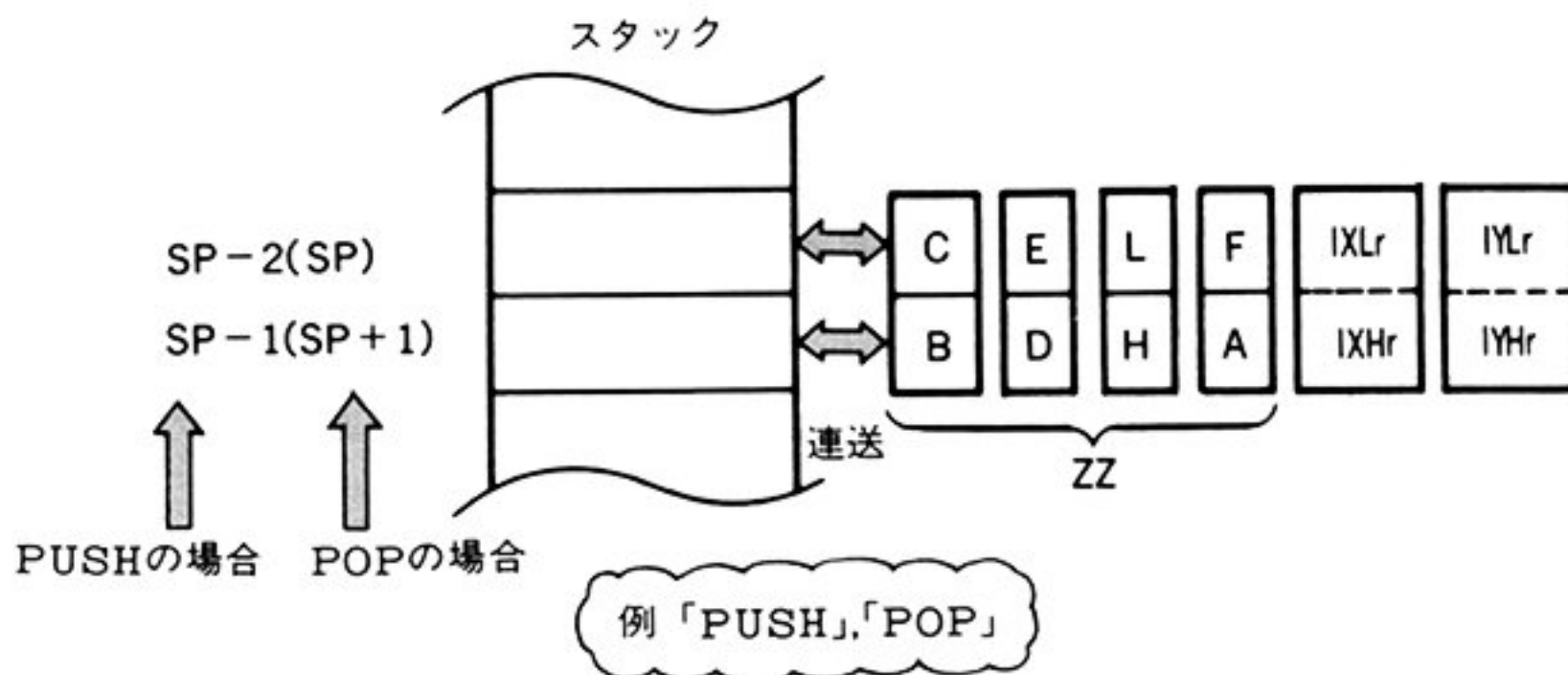



図2・7 スタックの内容とレジスタとの転送または交換

2・5 アドレシングモード

〔1〕 **インプライド**(IMP)　オペコード中に含まれる情報に基づくレジスタアドレシングモードです．たとえば，DAA，CPL，RLA命令などが相当します．

〔2〕 **レジスタダイレクト**(REG)　オペコード中の2ビットまたは3ビットで表されるフィールドによって，8ビット，16ビットのレジスタが指定されるアドレシングモードです．「LD B,C」,「INC B」,「MLT BC」命令などが相当します．

〔3〕 **レジスタインダイレクト**(REGI)　図2・8にこのアドレシングモードを示します．汎用レジスタを16ビットのアドレスレジスタとして使用し，これによってメモリ上のオペランドを指定するアドレシングモードです．「DEC(HL)」,「RL (HL)」,「LDD」命令などが相当します．

〔4〕 **インデックス**(INDX)　図2・9にこのアドレシングモードを示します．インデックスレジスタ（IX，IY）とディスプレースメント（d）との加算によって実効アドレスを生成するアドレシングモードです．「RL(IX+d)」,「DEC(IX+d)」,「SLA (IY+d)」命令などが相当します．

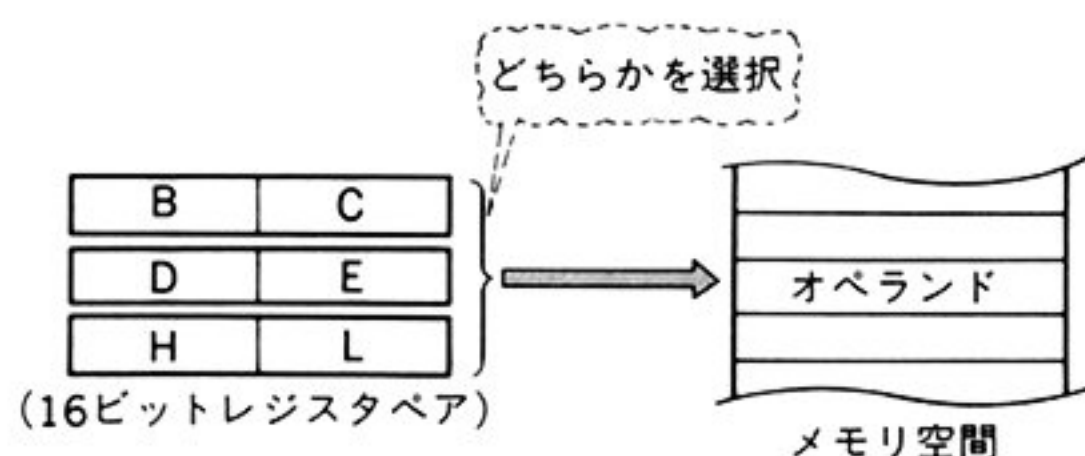



図 2・8 REGIアドレシング

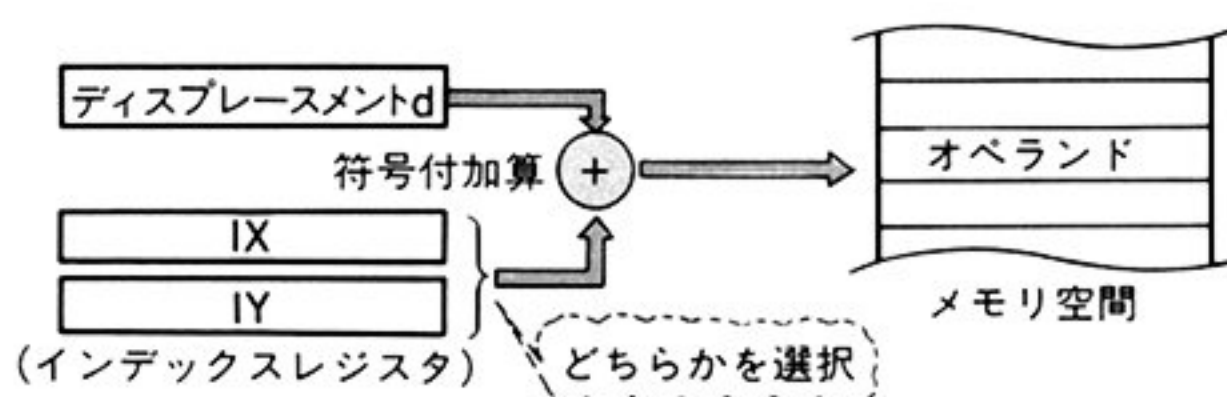



図 2・9 INDXアドレシング

〔5〕 **エクテンド(EXT)**　**図2・10**にこのアドレシングモードを示します．命令の2バイトを16ビットのアドレスとして，直接メモリ上のオペランドを指定するアドレシングモードです．「CALL」,「JP」命令などが相当します．

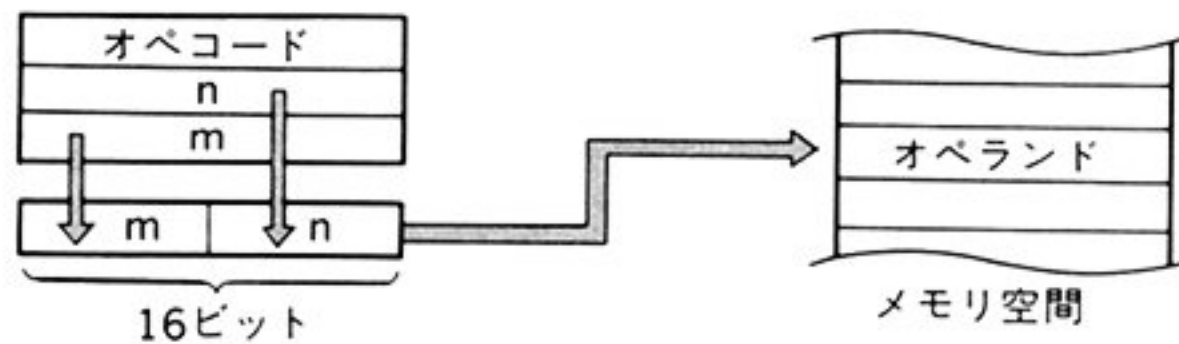



図 2・10　EXTアドレシング

〔6〕 **イミーディエイト(IMMED)**　**図2・11**にこのアドレシングモードを示します．命令中の1バイトまたは2バイトを直接オペランドとして使用するアドレシングモードです．「LD A, 00H」,「LD HL, 55AAH」などが相当します．

図 2・11　IMMEDアドレシング

〔7〕 **リラティブ(REL)**　**図2・12**にこのアドレシングモードを示します．プログラムカウンタにディスプレースメントを加算して，ブランチアドレスを生成するアドレシングモードです．「JR」,「DJNZ」命令などが相当します．

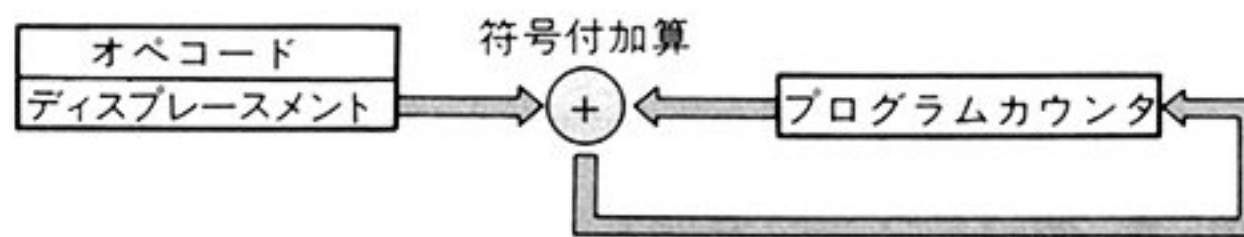



図 2・12　RELアドレシング

〔8〕 **IO**　I/O命令においてのみ使用されるアドレシングモードです．

2・6 メモリ空間とI/O空間

〔1〕 **概　　要**　64180では図2・13のように，データをアクセスする空間はメモリ空間とI/O空間の2つがあります．CPUが直接指定できるメモリ空間は$2^{16}=64$ Kバイトであり，I/O空間も64Kバイトです．また，64180はMMU (Memory Management Uuit) を内蔵しているので，MMUを通じてメモリ空間は最大1Mバイトに拡張できます．

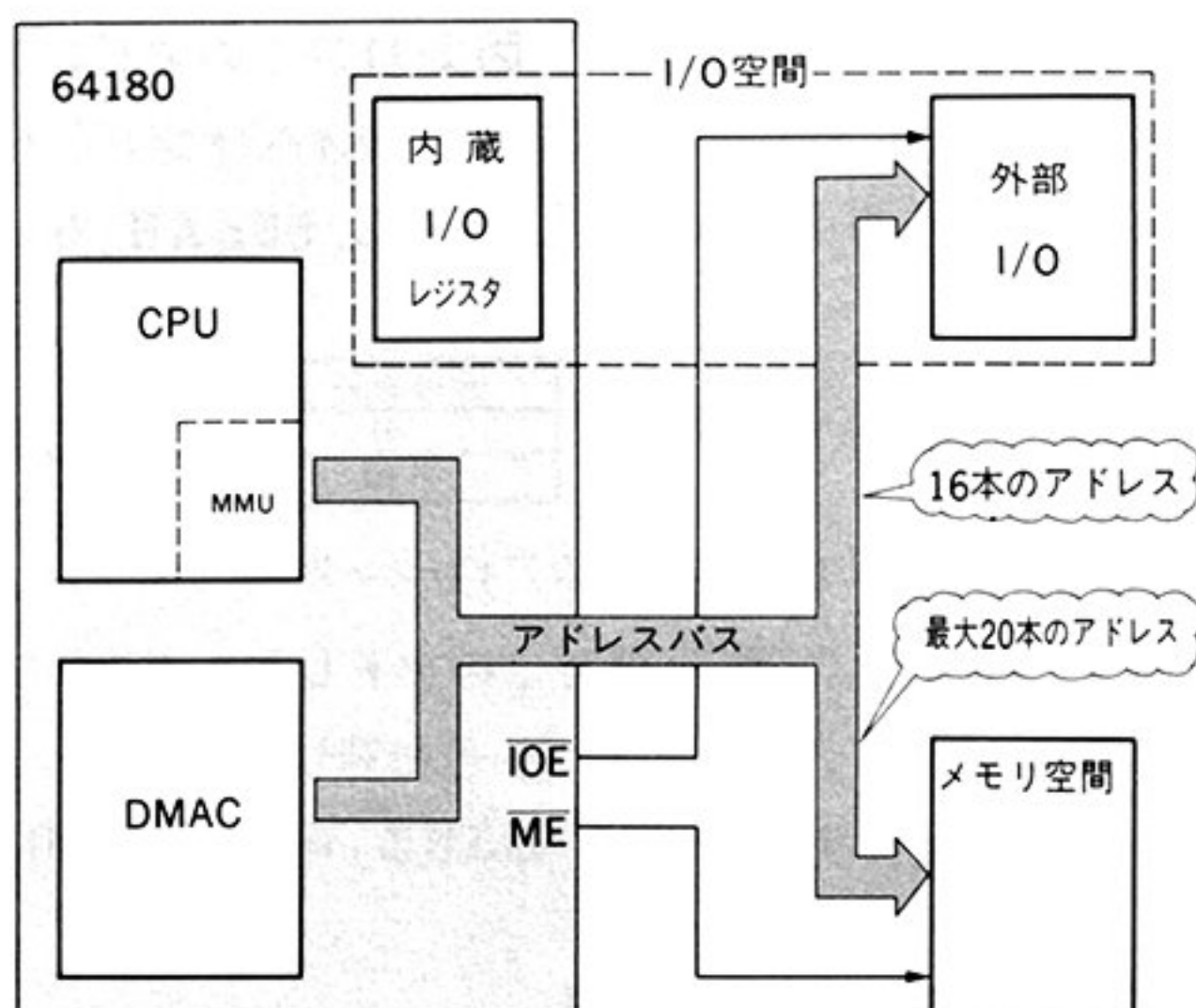



図 2・13　メモリ空間・I/O空間

〔2〕 **メ モ リ 空 間**　メモリ空間をアクセスする場合，制御信号の$\overline{\text{ME}}$が"L"になります．メモリ空間へのアクセスは，CPUまたはDMAC (Direct Memory Access Controller)が行います．CPUでは，「LD (HL), A」や「PUSH BC」「POP BC」命令などの通常の命令にて行われます．DMACでは，メモリ空間をアクセスするかI/O空間をアクセスするかを内蔵レジスタで設定し，メモリ空間が選択されると，CPUを介せずに直接アクセスされます．DMACがメモリ空間をアクセスする場合は，CPUと同じように$\overline{\text{ME}}$, $\overline{\text{RD}}$, $\overline{\text{WR}}$などの制御信号を出力します．

〔3〕 **I/O 空　　間**　I/O空間をアクセスする場合，制御信号の$\overline{\text{IOE}}$が

"L" になります．I/O 空間へのアクセスも，CPU または DMAC が行います．CPU では，I/O 空間をアクセスするための特別な命令（IN/OUT 命令）にて行います．DMAC では，内蔵 I/O レジスタで I/O 空間のアクセスを設定し，I/O 空間が選択されると CPU を介せずに直接アクセスされます．

I/O 空間の一部は図 2・13 の点線で示すように，内蔵の I/O レジスタがマッピングされています．そのアドレスについては付録を参照してください．なお，この内蔵 I/O レジスタのアドレスは上位 8 ビットは 00H に固定されており，下位 8 ビットは次のようになっています．ビット 0 からビット 5 は各レジスタに固有の値であり，ビット 6 とビット 7 は，I/O コントロールレジスタの IOA6 と IOA7 で設定します．したがって，**図 2・14** に示すように 64 バイト単位で 4 か所に移動できます．

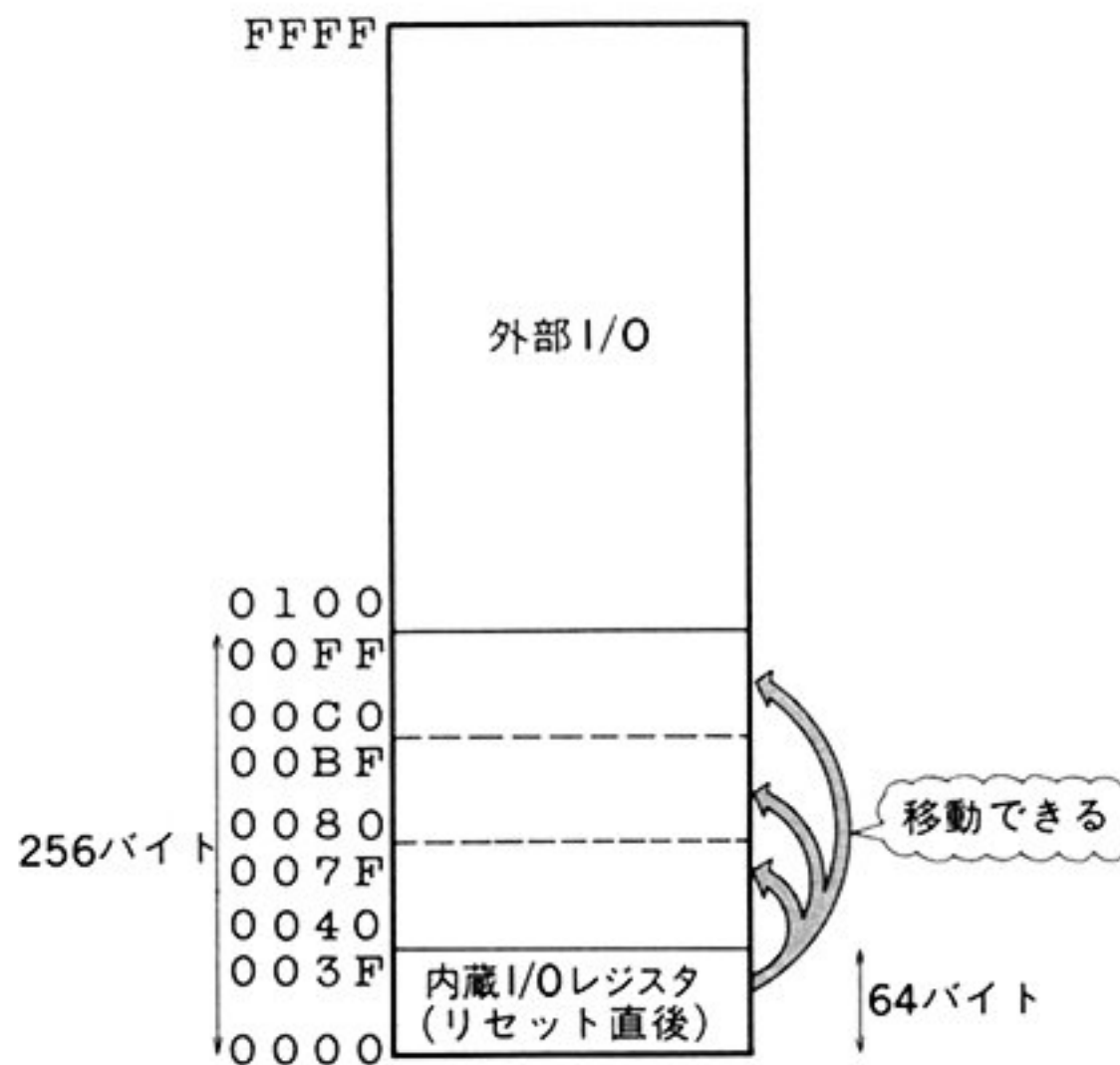



図 2・14　I/Oアドレスマップ

2・7 命令の分類

64180の命令は，**表2・4**に示す5種類に大別できます．

〔1〕 演算命令　メモリや8ビットまたは16ビットレジスタの数学的な論理演算を行う命令です．加算，減算，8ビット乗算，論理積，論理和，排他

表2・4 命令の分類

No.	大分類	中分類	小分類
1	演算命令	算術論理演算（8ビット）	ADD, ADC, SUB, SUBC, AND, OR, XOR, CP, INC, DEC, TST, CPL, MLT, NEG
		ローテートおよびシフト	RLA, RL, RLCA, RLC, RRA, RR, RRCA, RRC, SLA, SRA, SRL, RLD, RRD
		ビット操作	SET, RES, BIT
		算術演算（16ビット）	ADD, ADC, INC, DEC, SBC
2	転送命令	8ビット転送	レジスタ ←→ レジスタ レジスタ ←→ メモリ
		16ビット転送	レジスタ ←→ レジスタ レジスタ ←→ メモリ
		ブロック転送	LDD, LDI, LDDR, LDIR
		ブロックサーチ	CPD, CPI, CPDR, CPIR
		スタック処理	PUSH, POP
		交換	EX, EXX
3	プログラム制御命令	—	CALL, JP, JR, DJNZ, RET, RETI, RETN, RST
4	I/O命令	入力	IN, IN0, TSTIO
		出力	OUT, OUT0
		ブロック入力	IND, INI, INDR, INIR
		ブロック出力	OTDM, OUTI, OTIM, OUTD, OTDMR, OTDR, OTIR, OTIMR
5	特殊制御	演算	DAA, SCF, CCF
		割込み	EI, DI, IM
		動作モード	HALT, SLP
		その他	NOP

的論理和，インクリメント，デクリメント，コンペア，ローテートやシフト，ビットセットやリセット，ビットテストなどがあります．加算と減算，ローテートとシフトにはキャリーとともに実行する命令があります．

〔2〕 **転 送 命 令**　8ビットまたは16ビットのレジスタやメモリの内容を転送する最も基本的な命令です．単なるデータ転送のほかに，汎用レジスタをいくつか使って複数バイト転送するブロック転送命令やブロック比較命令，レジスタをスタックへ退避したり復帰したりする命令，レジスタとレジスタまたはスタックとの交換命令があります．

〔3〕 **プログラム制御命令**　プログラムの流れを制御する命令です．プログラムを分岐するジャンプ命令とサブルーチンコール命令，サブルーチンや割込み処理ルーチンからの復帰命令があります．また，特定のアドレスへ強制的に分岐するリスタート命令もあります．

〔4〕 **I/O 命 令**　I/Oとのデータの読出し・書込みを行う命令です．I/O空間の上位8ビットが常にOOHとした命令もあり，64180の内蔵I/Oレジスタのアクセスに使います．

〔5〕 **特殊制御命令**　1～4以外の命令で，10進補正，キャリー制御，割込み制御，およびホルトやスリープモード選択命令などがあります．

3. CPU タイミング

64180は「システム集積型」マイコンですが，ROMやRAMなどのメモリを内蔵していません．また，内蔵I/O機能だけでは不足する場合もあります．そのため，外部にメモリやI/O機器を接続します．そこで，64180はこれらを制御するための信号をもっており，そのタイミングは各種メモリや周辺LSIとのインタフェースが容易となるようにZ80から変更されています．Z80周辺LSIについては，64180Z版にてインタフェースが容易となるように工夫されています．

3・1 メモリリードサイクル・ライトサイクル

〔1〕 **概　　要**　メモリ空間へアクセスするときのサイクルで，1サイクルは T_1，T_2，T_3 ステートで構成されています．低速メモリのアクセスの場合は，ウェイトステートとして T_W が T_2 と T_3 の間に挿入されることがあります．

〔2〕 **メモリリードサイクル**　オペコードフェッチ以外のメモリリードサイクル（**図3･1**）です．そのシーケンスは次のとおりです．① T_1 ステートの前半でプログラムカウンタ PC の内容がアドレスバスに出力されます，② T_1 ステートの後半で，$\overline{ME}$ と $\overline{RD}$ がアクティブになりメモリへの起動がかかります，③データバス上にのったメモリリードデータを，T_3 ステートの ϕ クロックの立下りでCPUは取り込みます．そのとき，$\overline{ME}$ と $\overline{RD}$ は T_3 ステートの後半でインアクティブとなります，④アドレスバスは，次のサイクルの先頭までそのアドレスを保持しています．

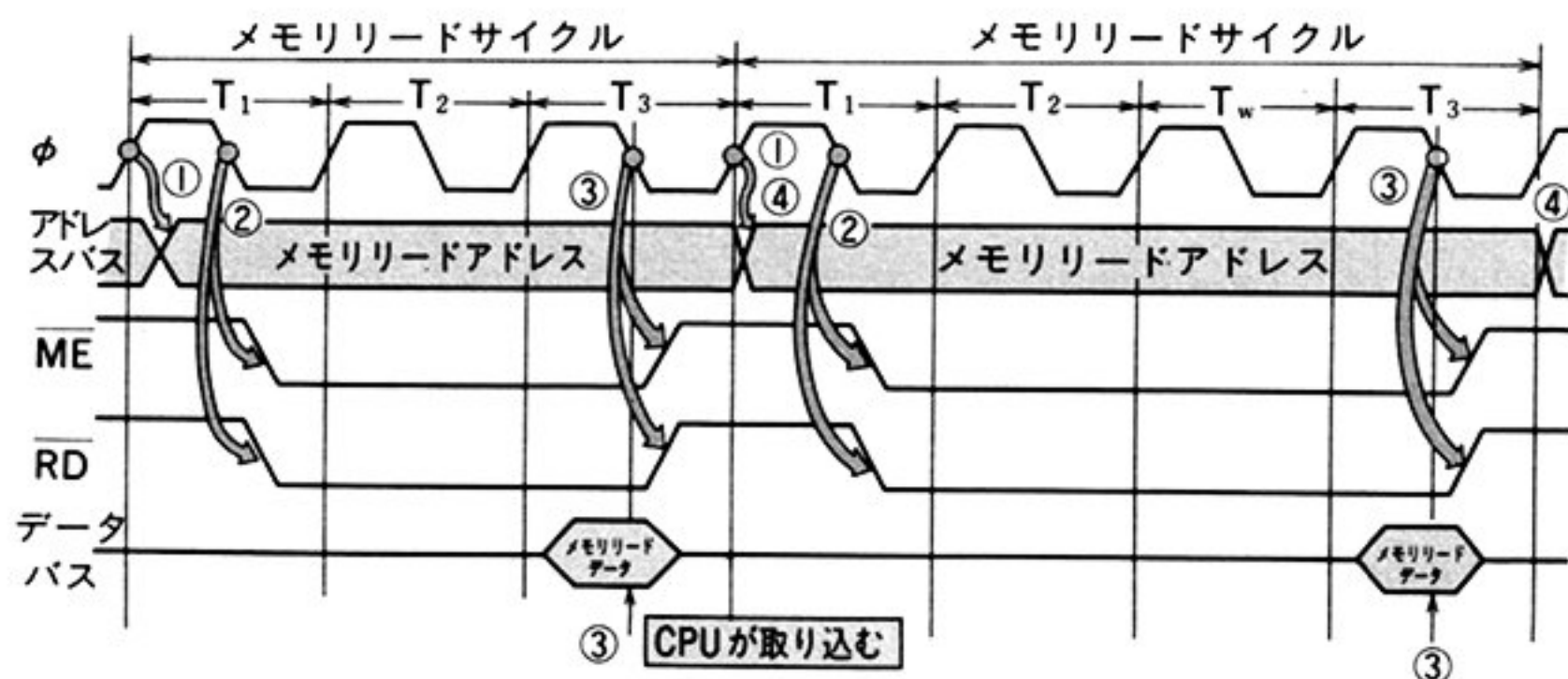



図 3・1　メモリリードサイクル

〔3〕 **メモリライトサイクル**　このサイクル（**図3･2**）のシーケンスは次のとおりです．①アドレスは T_1 ステートの前半で出力されます，② T_1 ステートの後半で，$\overline{ME}$ をアクティブにすると同時にメモリライトデータが出力されます，③ T_2 ステートの前半で $\overline{WR}$ がアクティブとなって，メモリに起動がかかります，④ $\overline{ME}$ と $\overline{WR}$ は T_3 ステートの後半でインアクティブとなりますが，メモリライトデータは次のバスサイクルの先頭まで保持されています．

Z80では，$\overline{WR}$ がアクティブになるタイミングは T_2 ステートの後半です．64180

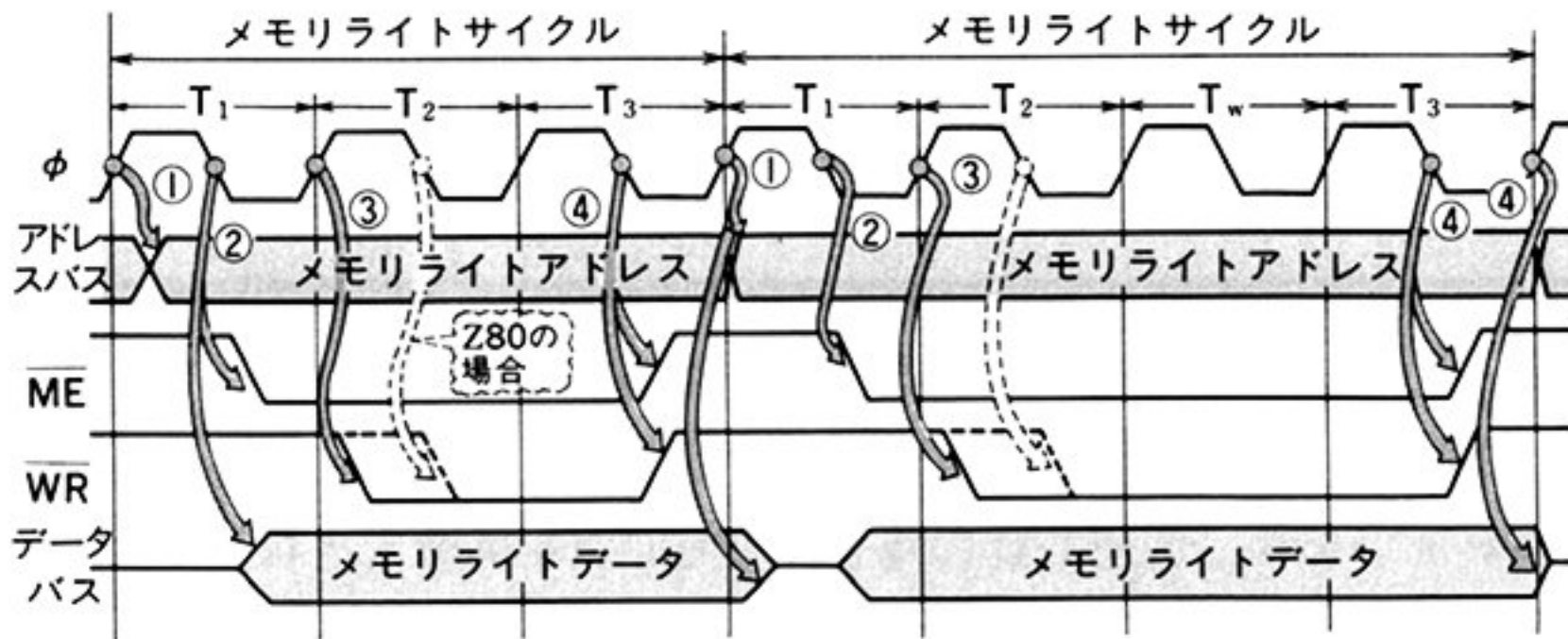



図 3・2 メモリライトサイクル

は，半クロック速くアクティブとなります．

〔4〕 **ウェイトステートが挿入されている場合** ウェイトステート T_Wは，T_2 ステートと全く同一動作をします．したがって，アドレス，$\overline{ME}$, $\overline{RD}$, $\overline{WR}$ などは，T_2 ステートから引き延ばされています．T_W ステートの挿入法は，端子要求によるものとソフトウェアによるものとがあります．

3・2 I/Oリードサイクル・ライトサイクル

〔1〕 **概　　要**　内蔵I/Oレジスタや外部I/O空間へのアクセス時のサイクルです．外部空間へのアクセス時では，1サイクルがT_1，T_2，T_W，T_3 ステートで構成されています．低速I/Oの場合，さらにT_Wが挿入されることがあります．

〔2〕 **I/Oリードサイクル**　このサイクル（**図3･3**）のシーケンスは次のとおりです．①T_1ステートの前半で，アドレスがアドレスバスに出力されます，②T_1ステートの後半で，$\overline{IOE}$と$\overline{RD}$がアクティブとなってI/Oに起動をかけます，③T_3ステートのϕクロックの立下りで，CPUはリードデータを取り込みます，それと同時に，$\overline{IOE}$と$\overline{RD}$はT_3ステートの後半でインアクティブとなり

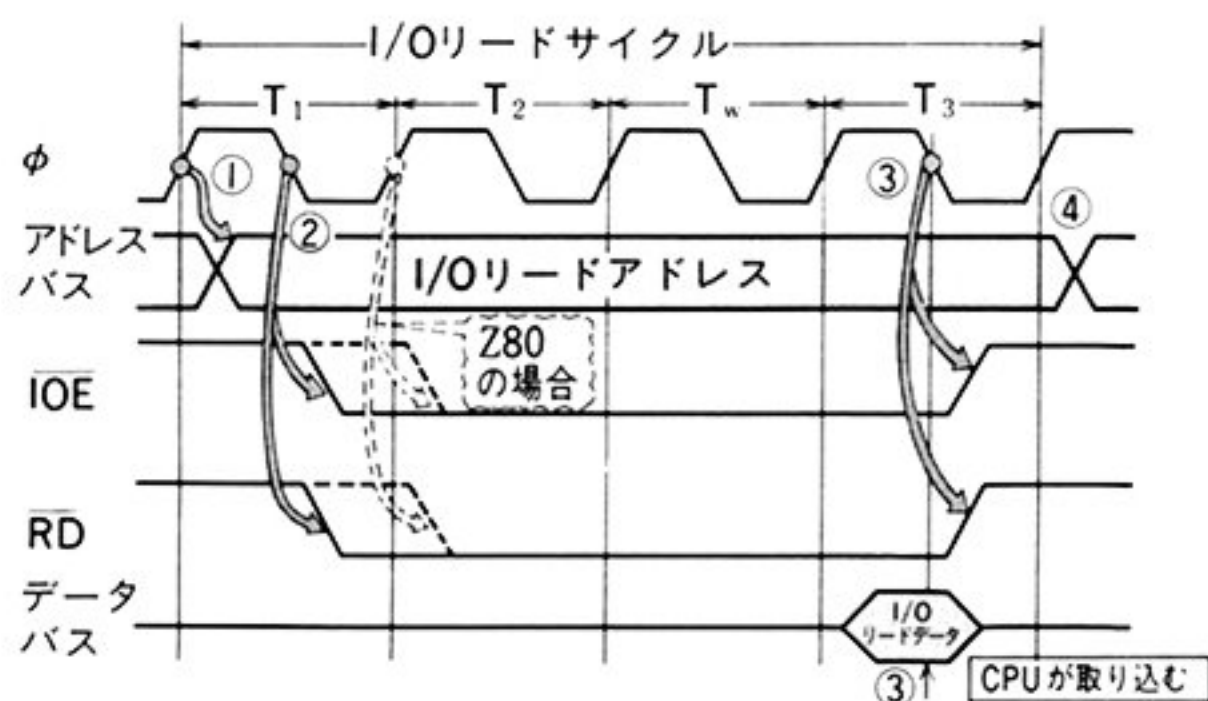



図 3・3　I/O リードサイクル

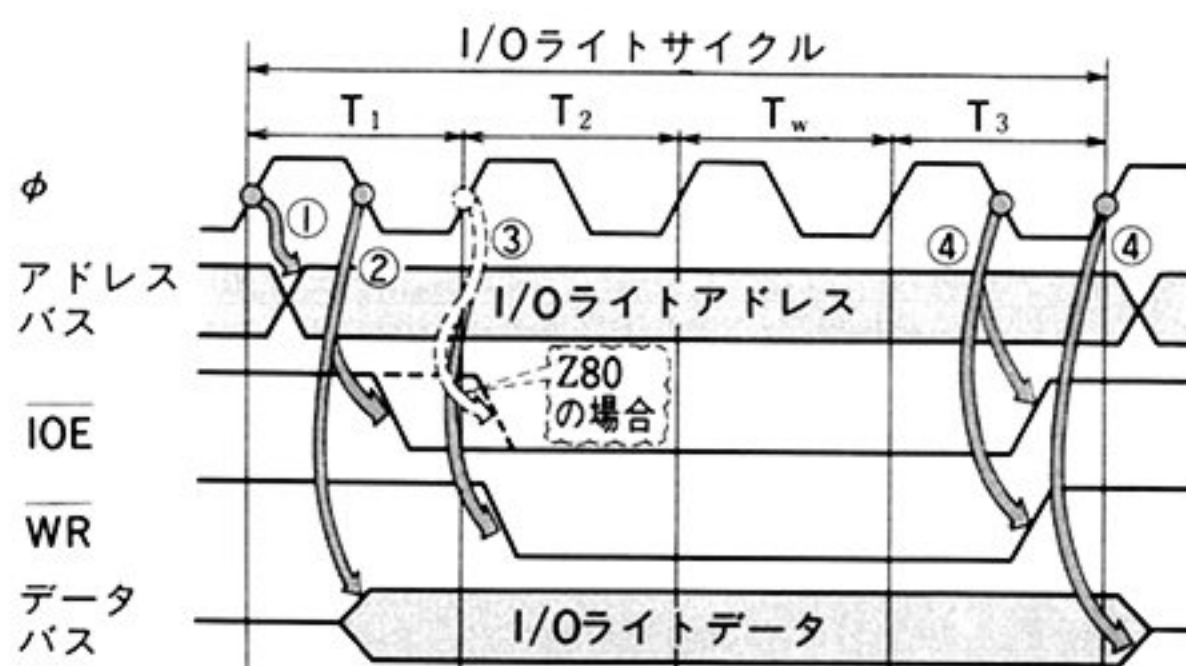



図 3・4　I/Oライトサイクル

ます，④アドレスは次のバスサイクルの先頭まで保持されています．

Z80では，$\overline{\mathrm{IORQ}}$と$\overline{\mathrm{RD}}$のアクティブタイミングはT_2ステートの前半ですので，64180は半クロック速くアクティブとなります．

〔3〕 **I/O ライトサイクル**　このサイクル（**図3・4**）のシーケンスは次のとおりです．①T_1ステートの前半で，アドレスがアドレスバスに出力されます，②T_1ステートの後半で$\overline{\mathrm{IOE}}$はアクティブとなり，同時にI/Oライトデータが出力されます，③T_2ステートの前半で$\overline{\mathrm{WR}}$はアクティブとなり，I/Oに起動がかかります，④ライトデータは次のバスサイクルまで保持されていますが，$\overline{\mathrm{IOE}}$と$\overline{\mathrm{WR}}$はT_3ステートの後半でインアクティブとなります．

Z80では，$\overline{\mathrm{IORQ}}$のアクティブタイミングはT_2ステートの前半ですので，64180は半クロック速くアクティブとなります．

〔4〕 **ウェイトステート挿入の場合**　これらのサイクルでも，メモリリード・ライトサイクル同様に，T_Wをさらに挿入することができます．

〔5〕 **64180Z版の場合**　$\overline{\mathrm{IOE}}$と$\overline{\mathrm{RD}}$のアクティブタイミングを，ソフトウェアでT_2の前半に変更することができます．内蔵の動作モードコントロールレジスタの$\overline{\mathrm{IOC}}$ビットを“0”に設定します．Z80周辺LSIとのインタフェースに便利です（**図3・5**，**図3・6**）．

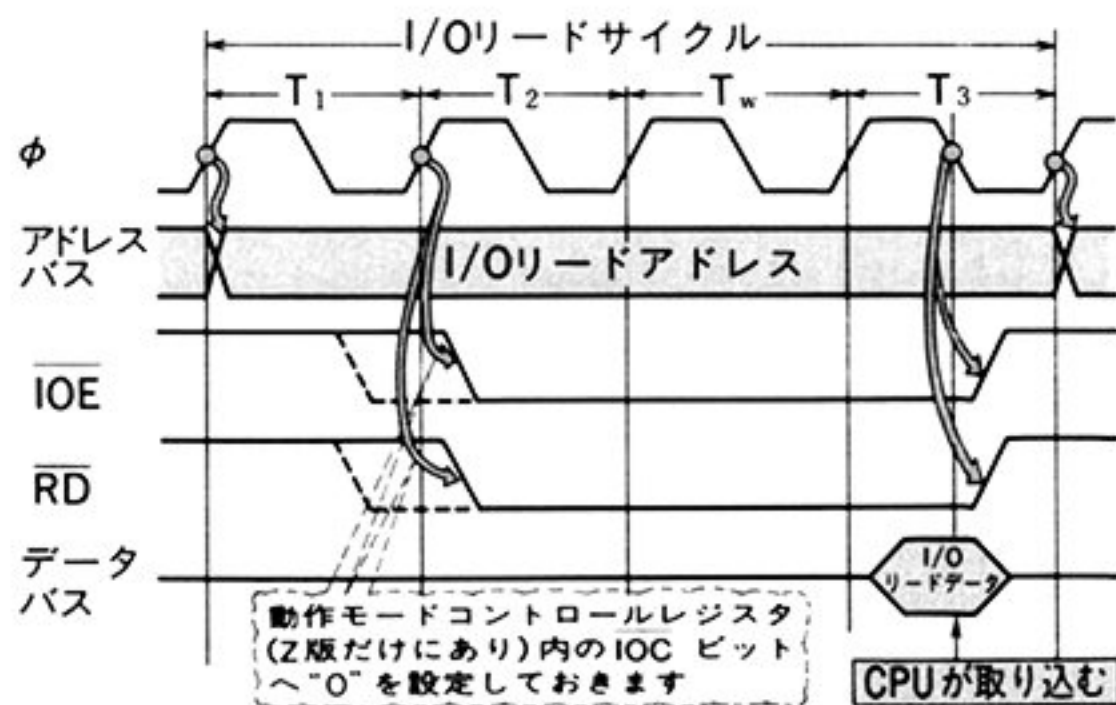



図3・5 I/Oリードサイクル（$\overline{\mathrm{IOC}}$＝“0”の場合）

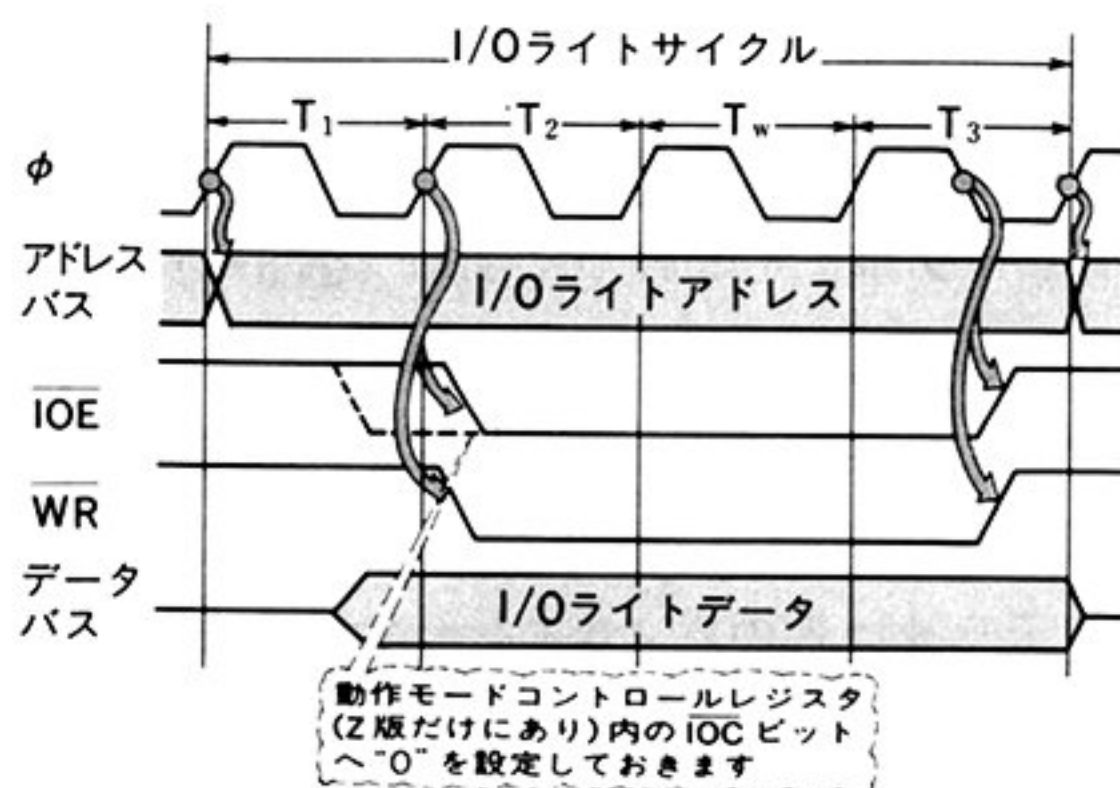



図 3・6 I/Oライトサイクル ($\overline{\text{IOC}}$ ＝"0"の場合)

64180Z版

64180Z（Z180）では，Z80周辺LSIとのインタフェースが容易になります．64180R1版から，$\overline{\text{LIR}}$，$\overline{\text{IOE}}$，$\overline{\text{RD}}$タイミングとRETI命令の実行タイミングが改善されています．Z80周辺LSIを使用するシステムには，64180Zが適しているでしょう．

3・3 オペコードフェッチサイクルと命令の実行

〔1〕 **オペコードフェッチサイクル**　このサイクル（**図3・7**）は，メモリリードサイクルと同様に T_1，T_2，T_3 から構成されていますが，次の点で異なります．

① $\overline{LIR}$ 信号が“L”になります．

② リードデータは，半クロック速く確定しなければなりません．T_3 ステートの ϕ クロックの立上りで，CPU はオペコードを取り込みます．

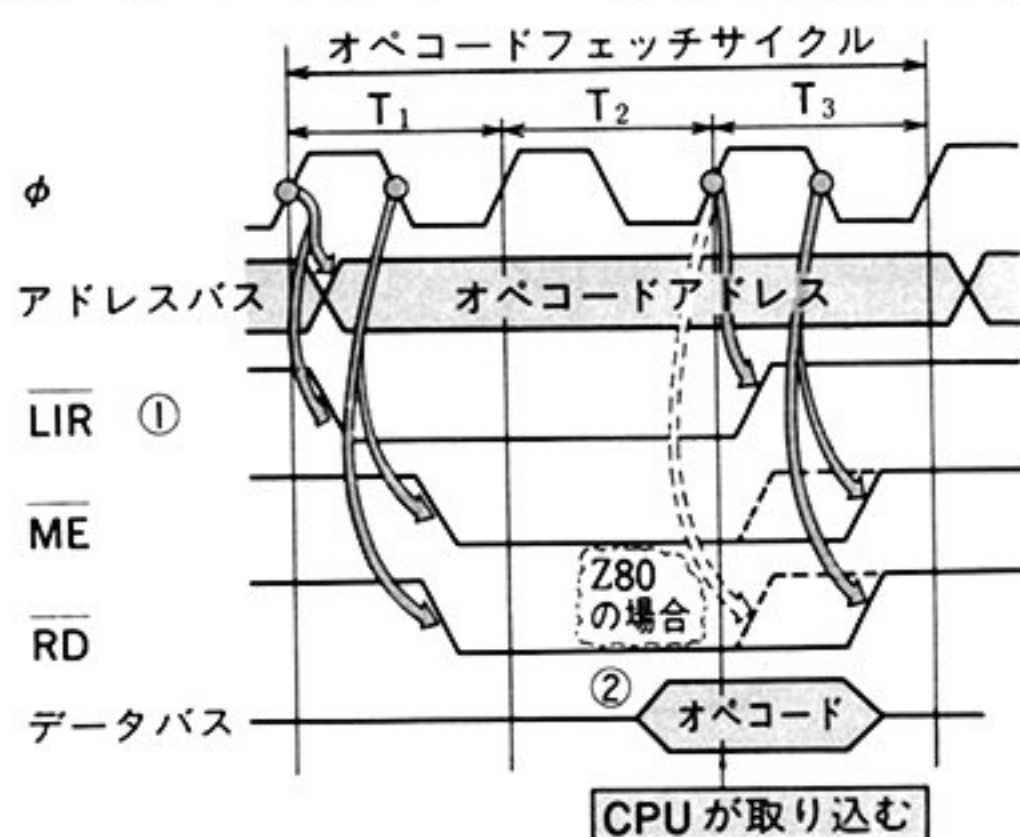



図 3・7　オペコードフェッチサイクル

また，オペコードフェッチサイクルでは，そのオペコードが1バイト目のオペコードであることを表すために，ST 信号が“L”になります．タイミングは $\overline{ME}$ と同じです．

〔2〕 **Z80 との 相 違**　Z80 では1サイクルが T_1, T_2, T_3, T_4 で構成されており，T_3 と T_4 はリフレッシュサイクルとも呼ばれています．64180 では，リフレッシュサイクルはオペコードフェッチサイクルと独立しており，非同期に入ります．また，Z80 では $\overline{MREQ}$ と $\overline{RD}$ のインアクティブタイミングは T_3 の前半なので，64180 は半クロック遅くインアクティブとなります．

〔3〕 **命 令 の 実 行**　CPU の基本動作は，1個または複数個のマシンサイクル（MC）によって構成されています．1マシンサイクルは，データのリード・ライトが伴う場合では T_1，T_2，T_3 の3ステートまたは T_1，T_2，T_W，T_3 の4ス

テートですが，データのリードライトが伴わない場合では，1マシンサイクルが1ステートです．なお，システムに使用しているメモリまたはI/Oのアクセスタイムが長い場合は，T_3 の前に T_W を挿入してマシンサイクルを引き延ばします．

〔4〕 **命令の実行タイミング例**　I/O命令OUTO (m), Aの例を示します．この命令は，2つのオペコードフェッチと1つのオペランドリード，さらにI/OライトとCPU内部状態から成ります．図3・8の例では，メモリ空間をアクセス時 T_W が挿入されない場合を示しています．

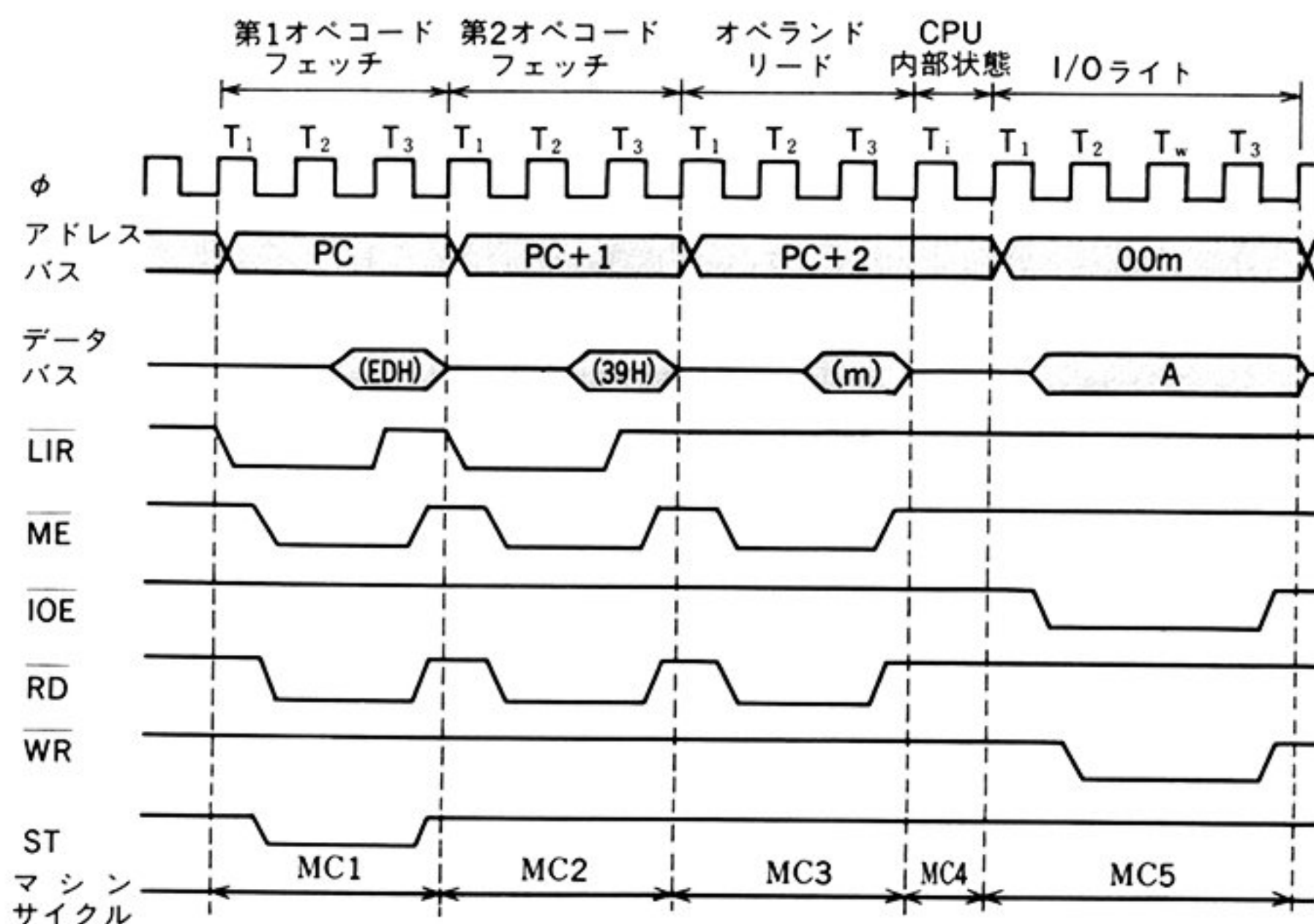



図 3・8　基本命令実行タイミング(OUTO(m)，Aの場合)

4. メモリマネジメントユニット

64180は，8ビットのCPUで世界で初めて1Mバイトのアドレス空間をサポートしたMMU（メモリマネジメントユニット）を内蔵しています．従来の8ビットCPUはプログラムエリアが64Kバイトという制約があり，多くの応用機器においてアドレス空間が不足していました．64180は，こうした大容量のメモリを必要とする応用機器で使用されることが多いバンクスイッチ方式のMMUを内蔵しています．

4・1 MMUの概念

〔1〕 **64180内蔵MMUの機能**　Z80をはじめ，8ビットCPUは16ビットのアドレスをもっています．したがって8ビットCPUは，接続できるメモリが2^{16}→64Kバイトとなります．

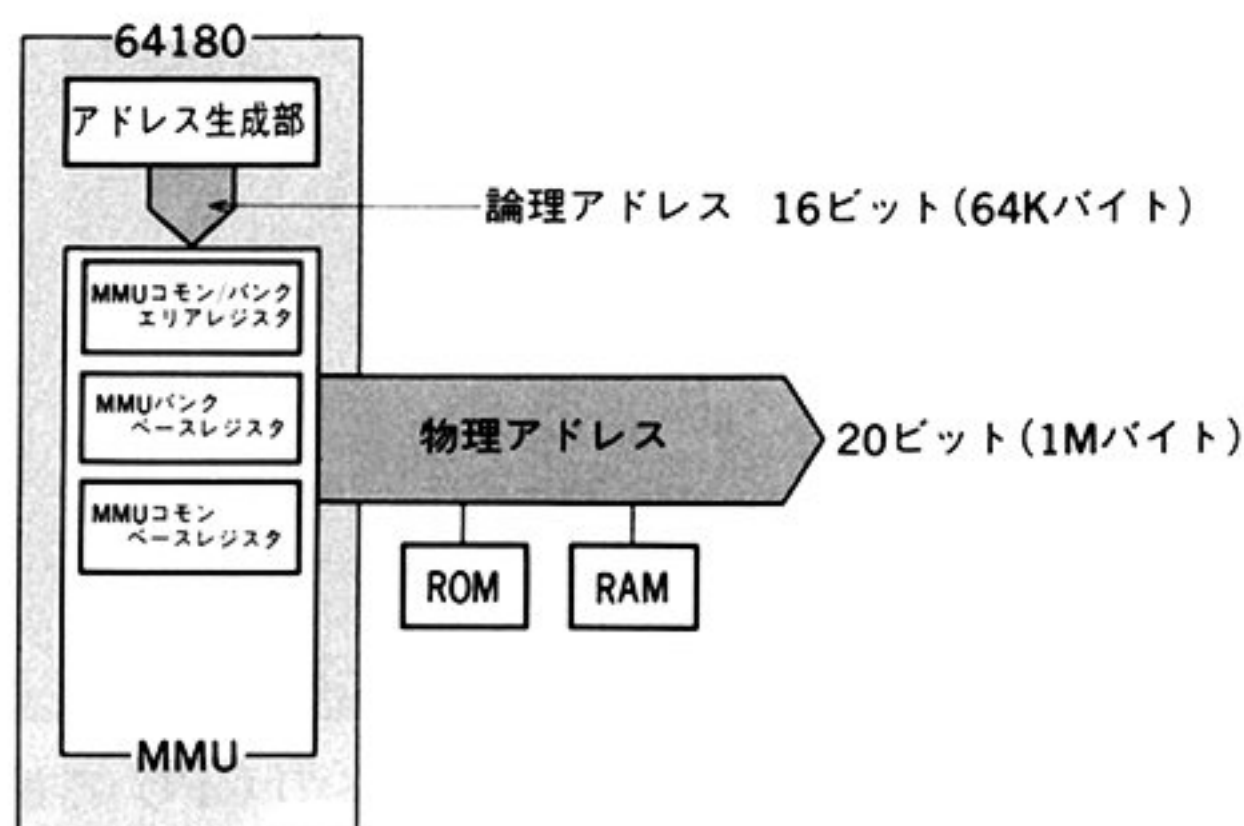
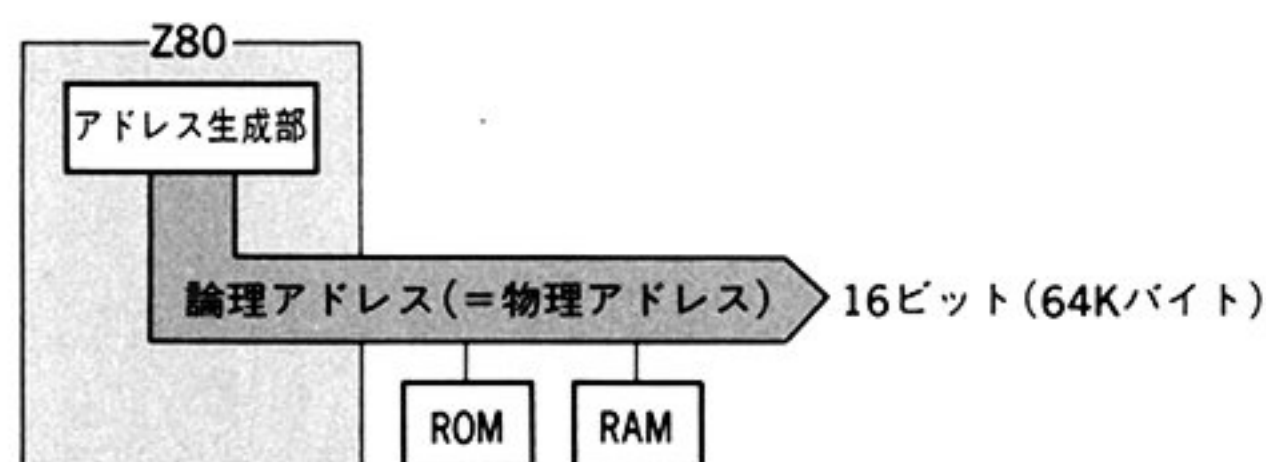



図 4・1　64180のMMUの機能と構成

一方，現在多くの応用分野において，64Kバイトではメモリ空間が不足することが多くなっています．64180の**メモリマネジメントユニット**(Memory Management Unit)は，バンクスイッチにより64Kバイトを超える大容量のメモリを接続可能とします．具体的には，64180のMMUは最大20ビットの物理アドレスを生成します．すなわち，64180は2^{20}→1Mバイトのメモリ空間をもつことができます（図4・1）．

〔2〕 **64180内蔵MMUの構成**　64180のMMUは，16ビットの論理アドレ

スを20ビットの物理アドレスに変換するため，3つのレジスタをもっています．これらのレジスタは，MMUコモン/バンクエリアレジスタ，MMUバンクベースレジスタ，MMUコモンベースレジスタと呼ばれ，I/O空間上に配置されています．64180のMMUは，この3つのレジスタと制御，演算回路により構成されています．

〔3〕 **MMUの動作範囲** 64180のMMUは，CPUがメモリをアクセスする場合に動作します．これは以下に示す4通りの場合です．

(1) オペコードフェッチ

(2) メモリへのデータのリード/ライト

(3) 割込みベクタリングアドレス生成

(4) 割込みのリスタートアドレス生成

同じメモリへのリード/ライトでも，内蔵のDMAC動作中はMMUはバイパスされ，内蔵DMACから直接物理アドレスが出力されます．また，I/O空間へのリード/ライトの場合もMMUはバイパスされます．この場合，A_{15}～A_0には有効なI/Oアドレスが出力されますが，A_{19}～A_{16}にはOOHが出力されます．

4・2 MMUの動作

4・2・1 エリアレジスタの機能

〔1〕 **論理空間の分割** 64180のMMUは，64Kバイトの論理アドレス空間を最大3つのエリア（コモンエリア0，バンクエリア，コモンエリア1）に分割します．この分割の指定を行うのが，**MMUコモンバンクエリアレジスタ**（以下エリアレジスタと略す）です．

このエリアレジスタは8ビットから成り，上位4ビットでコモンエリア1の下限を，下位4ビットでバンクエリアの下限を指定します．エリアレジスタの構成と機能を**図4・2**に示します．これら4ビットの指定は，おのおのの空間の最上位4ビットのアドレス（A_{15}～A_{12}）に対し有効となります．

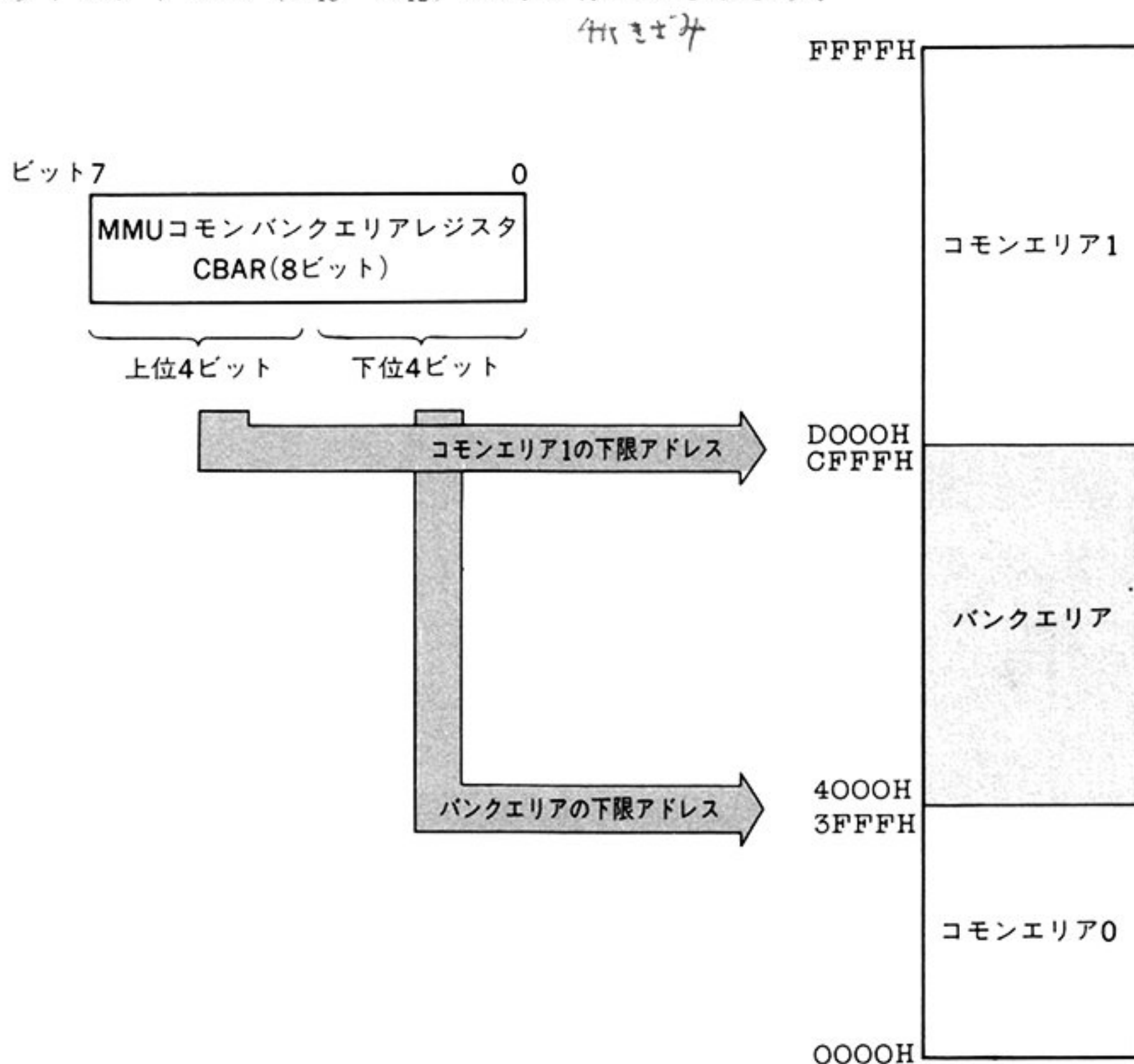



図 4・2 論理空間の分割

〔2〕 **エリアレジスタの設定例** たとえば，エリアレジスタに 'D4H' の設定を行うと，コモンエリア1の下限の指定は 'D' となり，D000HからFFFFHがコモンエリア1となります．同様に，バンクエリアの下限の指定は '4' となり，4000HからCFFFHがバンクエリアとなります．コモンエリア0の下限は常に0であるため，0000H〜3FFFHがコモンエリア0となります（図4・3）．

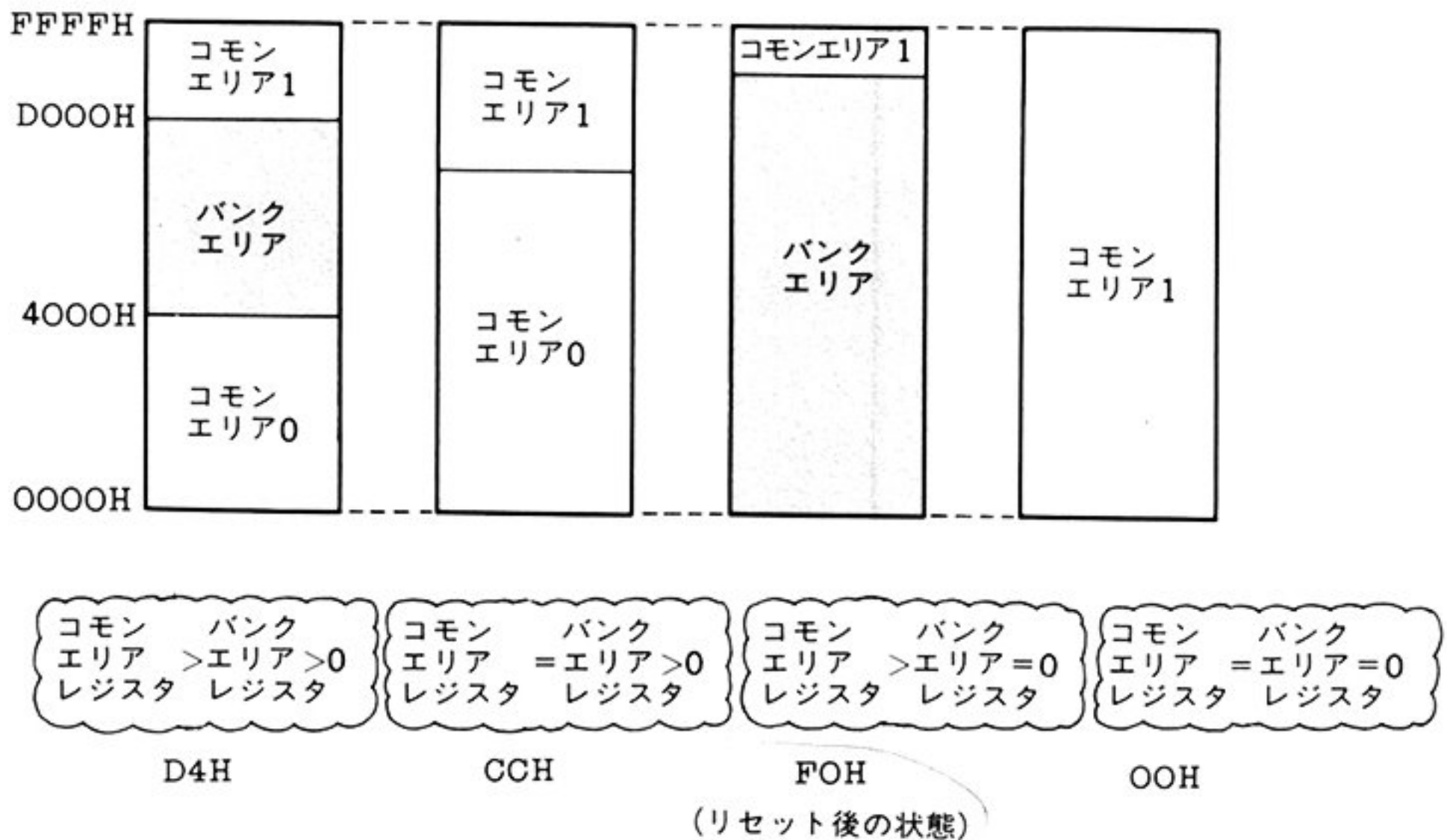



図 4・3 エリアレジスタの設定例

これ以外にも，コモンエリア1の下限＝バンクエリアの下限（＞0）の設定により，コモンエリア1とコモンエリア0により2分割の構成をとったり，コモンエリア1の下限＞バンクエリアの下限＝0の設定により，コモンエリア1とバンクエリアにより2分割の構成をとったり，コモンエリア1の下限＝バンクエリアの下限＝0の設定により，コモンエリア1により論理空間を構成することもできます．このように，エリアレジスタの設定により4通りの論理アドレス空間の組合せのうち1つを選び，各エリアの空間の大きさを指定することができます．

なお，エリアレジスタはリセット直後F0Hに初期化されます．

コモンエリア1の下限アドレスとバンクエリアの下限アドレスは，常にコモンエリア1の下限アドレスが大きくなるように設定する必要があります．すなわちバンクエリア≧0となる必要があります．

4・2・2 ベースレジスタの機能

〔1〕 ベースレジスタ ベースレジスタ(**MMUコモンベースレジスタ, MMU**

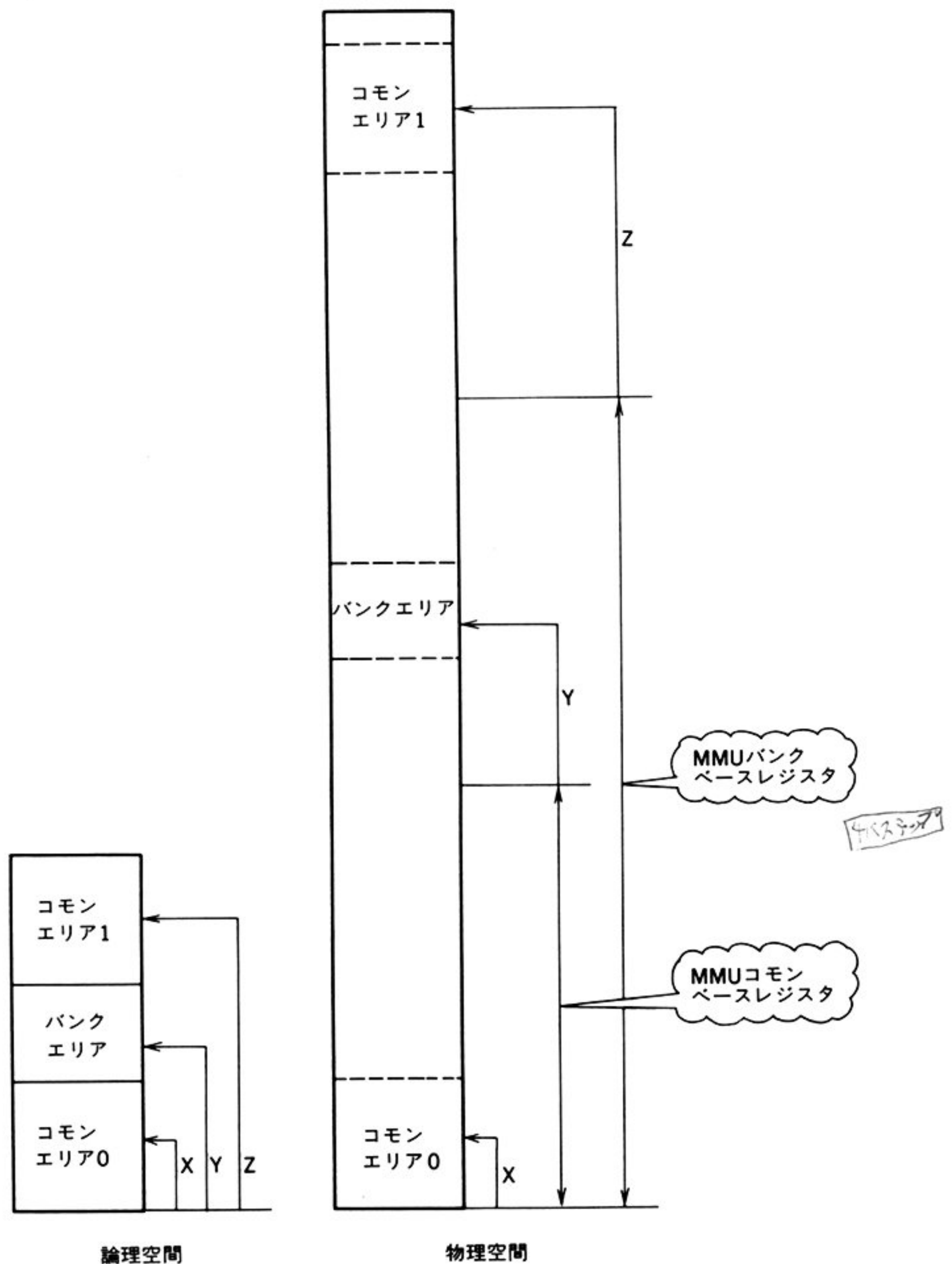



図 4・4 論理空間から物理空間への写像

バンクベースレジスタを総称してベースレジスタと呼び，以下この2つのレジスタをベースレジスタと略す）は，エリアレジスタによって分割された3つのエリアのうち，コモンエリア1とバンクエリアに対しオフセット値を与えるレジスタです．すなわち，この2つのエリアの論理アドレスに対してはベースレジスタの値が加算され，実際にアクセスされる物理アドレスが決定されます．それぞれコモンエリア1に対してはコモンベースレジスタ，バンクエリアに対してはバンクベースレジスタが対応しています（図4•4）．

〔2〕 **物理アドレスの生成**　たとえば，バンクエリア上のメモリ（論理アドレスで指定）をアクセスしたとします．このときの論理アドレスとバンクベースレジスタの値が加算され，物理アドレスが生成されます．実際の加算は次のように行われます．論理アドレスの最上位の4ビット（LA_{15}～LA_{12}）とベースレジ

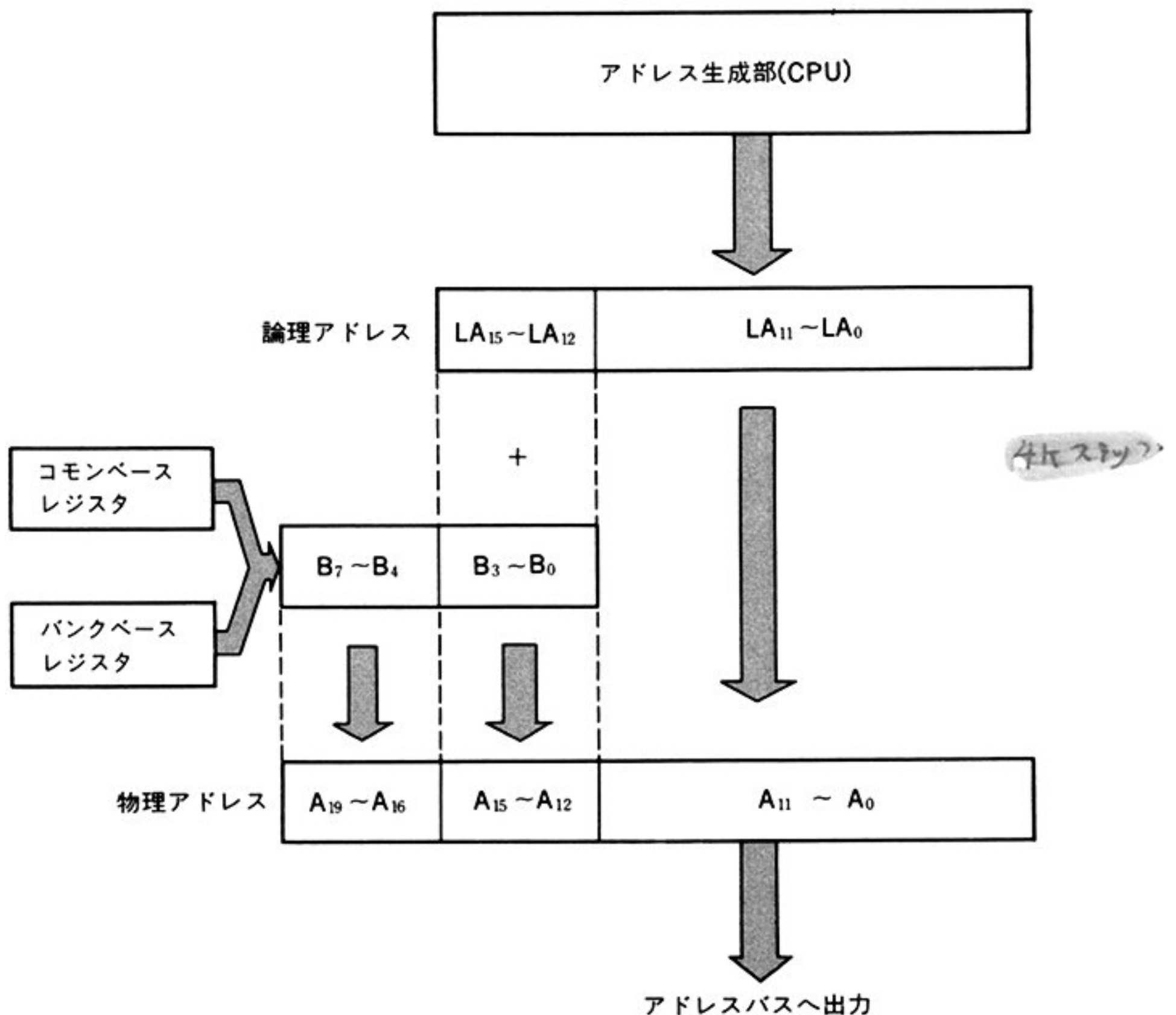



図 4・5　物理アドレスの生成

スタの下位4ビット（B_3～B_0）が加算され，物理アドレスのA_{15}～A_{12}が生成されます．また，この加算の際のキャリーとベースレジスタの上位4ビット（B_7～B_4）が加算され，物理アドレスのA_{19}～A_{16}に出力されます．論理アドレスの下位12ビット（LA_{11}～LA_0）はスルーで物理アドレスのA_{11}～A_0となります．この物理アドレスの生成の様子を図4•5に示します．

これはコモンエリア1についても全く同様で，この場合のオフセット値としてはコモンベースレジスタが使用されます．ただし，コモンエリア0についてはオフセット値は常に'0'となります．すなわち，コモンエリア0は常に論理アドレス＝物理アドレスであり，物理空間上の0番地から配置されます．

なおベースレジスタは，リセット直後コモンベースレジスタ，バンクベースレレジスタとも00Hに初期化されます．したがって，この状態では論理空間＝物理空間になっており，Z80同様64Kバイトの論理アドレス空間をもつCPUに見えます．

4・3 MMUの使用例

4・3・1 ROMと256KDRAMのアクセス

〔1〕 **マッピングの検討** 64180のMMUを使用し，32Kバイトの EPROM（27256）と256KバイトのRAM（DRAM 日立HM51256など）をアクセスする方法を考えてみましょう（図4•6）．

プログラムエリアとして64Kバイトの論理アドレス空間のうち32Kバイトを ROMに割り当てます．これでプログラムは論理空間上に常駐していることになります．残りの32KバイトをRAMエリアとして割り当てます．ここでスタックなどのワークエリアを設けるため，4Kバイトだけはコモンエリア1に割り当てます．コモンベースレジスタは固定とし，論理空間上の最上位に常駐させます．残った28Kバイトはバンクエリアに割り当て，バンク切替えを行ってデータメモリエリアとして使用します．

〔2〕 **エリアレジスタの設定** 前述のマッピングを実現するため，エリアレジスタにF8Hを設定します．この設定によりワークエリアのためのコモンエリア1がF000HからFFFFH，データメモリのためのバンクエリアが8000HからEFFFH，プログラムエリアのためのコモンエリア0が0000Hから7FFFHとなります．

〔3〕 **ベースレジスタの設定** 256KDRAMは，64180の1Mバイトのアドレス空間のC0000HからFFFFFHに割りつけます．このうちFF000H～FFFFFHの4KBをコモンエリア1に割り当て，ワークエリアとして使用します．コモンベースレジスタにはF0Hを設定します．この値は固定とし書き換える必要はありません．DRAM の残りの252Kバイトを28Kバイトのバンクエリアのバンク切替えでアクセスします．この場合9個(252÷28)のバンクをもつことができます．その分割は，バンクエリア(1)はC0000H～C6FFFH，バンクエリア(2)はC7000H～CDFFFHなどとなります．たとえば，バンクエリア(1)をアクセスしようとした場合，バンクベースレジスタにB8Hを設定することになります．

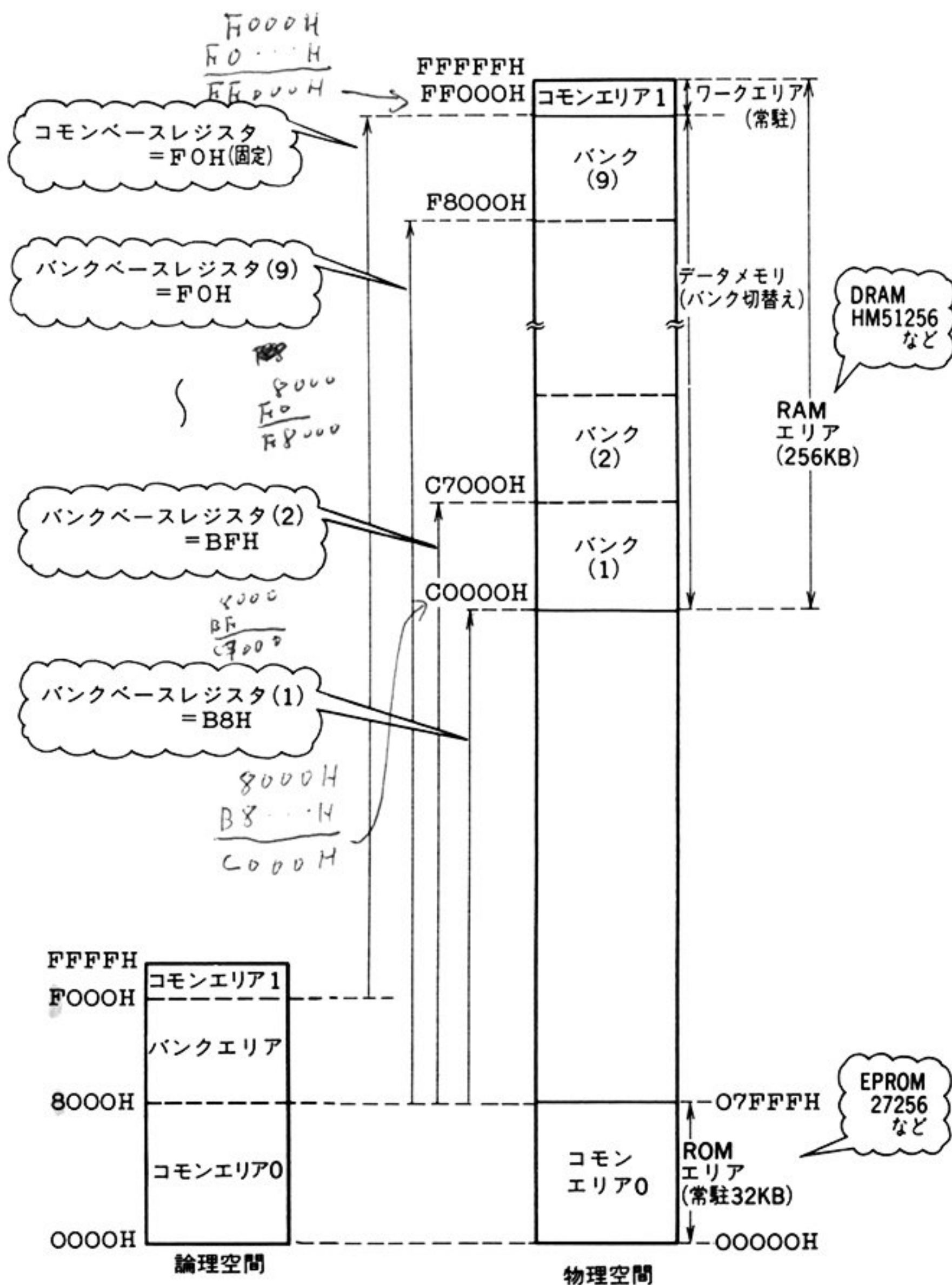



図 4・6　32Kバイト ROM と 256Kバイト RAM のアクセス

4・3・2 1Mバイトを超えたアドレスのアクセス

コモンエリア1によるコモンエリア0のアクセス　64180のMMUでは，論理アドレスとベースレジスタの和が20ビットのアドレス幅を超える場合があります．たとえば，コモンエリア1がF000H～FFFFHの場合，コモンベースレジスタにFFHを設定したとします．この結果は，コモンエリア1に相当する物理アドレスが10E000H～10EFFFHとなり，20ビットの物理アドレス幅を超えています．この場合，オーバフローした21ビット目の値は無視され，実際の物理アドレスとしては0E000H～0EFFFHが出力されます．すなわち，FFFFFHを超えた物理アドレスが生成されるとキャリーが無視され，00000Hからアクセスされることになります．

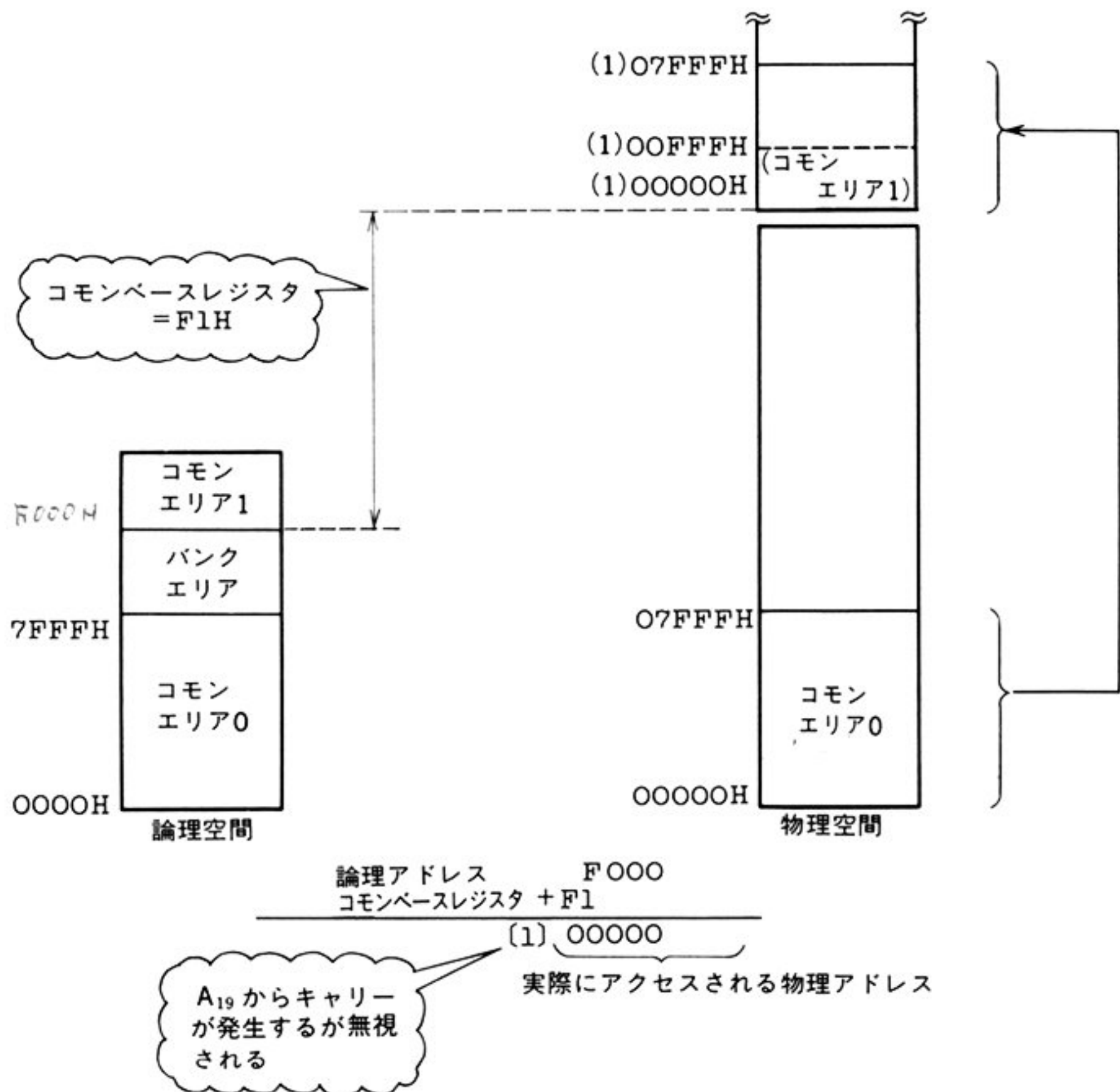



図 4・7 1Mバイトを超えたアドレスのアクセス

これを利用して，コモンエリア1のウィンドーからコモンエリア0に相当する論理アドレスをアクセスすることができます．たとえば，コモンベースレジスタにF0Hを設定することにより，コモンエリア0の0000H～0FFFH がアクセスされ，F1H を設定すると 1000H～1FFFH がアクセスされます（図4•7）.

ベースレジスタ設定時に自分自身を制御するベースレジスタを変更すると，プログラムが複雑になるため，注意が必要です．

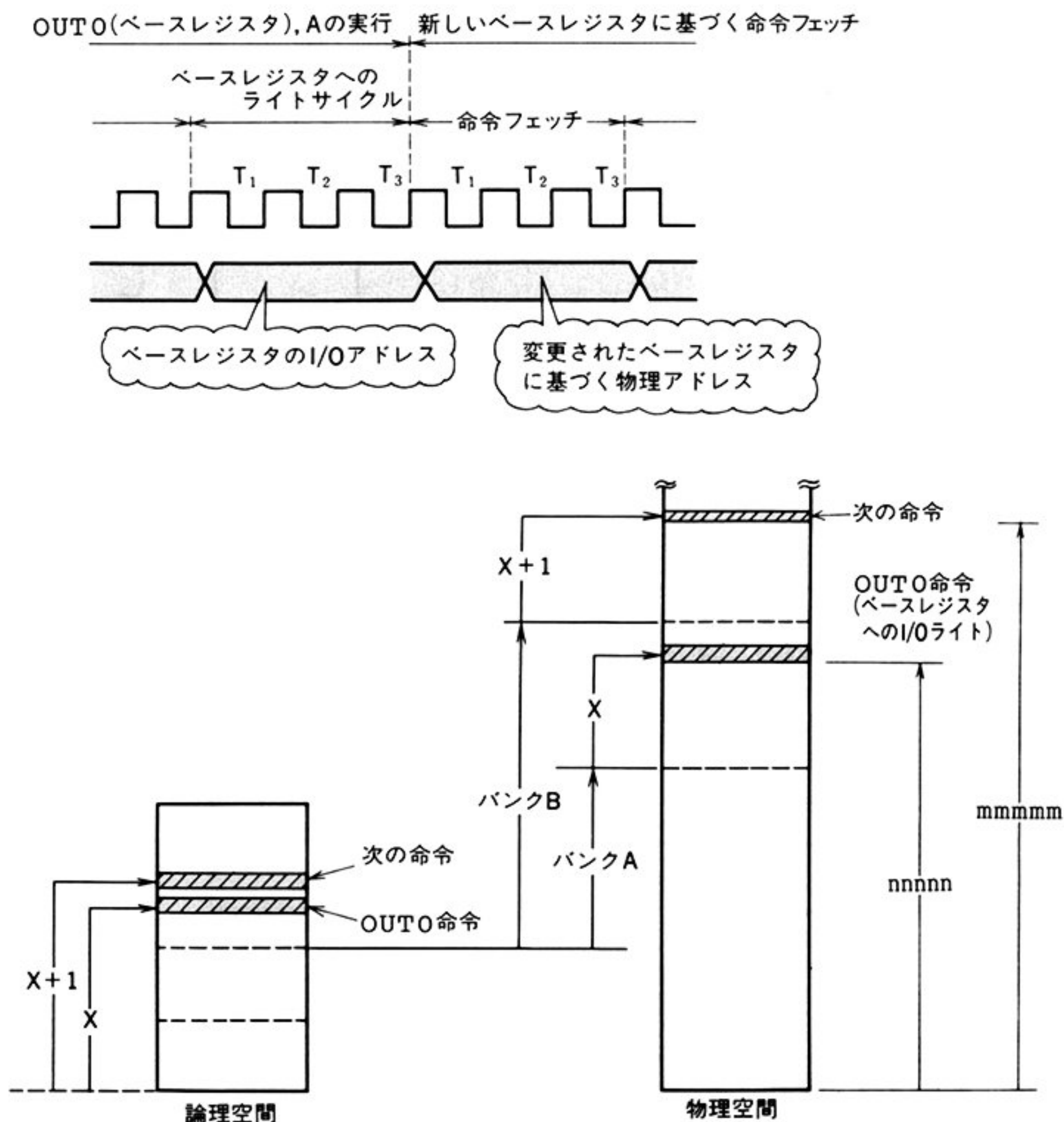



図 4・8 ベースレジスタの設定

たとえば，コモンベースレジスタをコモンエリア1から変更した場合です．ベースレジスタの変更を行うと，ベースレジスタへのI/Oライトサイクルの直後から，新しいベースレジスタの値に基づいた物理アドレスからオペコードをフェッチします．したがって，ベースレジスタへのライト命令〔OUTO(m), Aなど〕がコモンエリア1（バンクA）の nnnnn 番地にあると，次はコモンエリア1（バンクB）の mmmmm 番地からフェッチが行われます．したがって，コモンエリア1からコモンベースレジスタを変更する場合，プログラムの配置に工夫が必要です（図4•8）．

こうした手間を避けるため，ベースレジスタを変更する場合，バンクベースレジスタの書換えはコモンエリア0またはコモンエリア1，コモンベースレジスタの書換えはコモンエリア0かバンクエリアから行うのが便利です．

第Ⅱ編

周辺機能

システムインテグレーションとは，多くの周辺機能を1チップ上に内蔵することであり，64180はタイマ，非同期方式シリアルI/O，クロック同期方式シリアルI/O, WAITコントローラ, DRAMリフレッシュコントローラ，DMAコントローラなど豊富な周辺機能をオンチップしており，究極の8ビットマイクロコンピュータと呼ばれています．特に，2チャネルの非同期方式シリアルI/Oと2チャネルのDMAコントローラは，多くのシステムのプリント基板サイズを小さくしました．

5. タ イ マ

NCやプリンタなどの多くのアプリケーションでは，タイムベースやパルス出力などを制御するタイマが必要になることがあります．64180では，16ビットデータ長のプログラマブルリロードタイマを2チャネル内蔵しています．そのうち，チャネル1にはタイマ出力端子をもっています．これらのタイマを使用することにより，ソフトウェアの負担が軽減できます．

5・1　タイマの構造

64180にはプログラマブルリロードタイマが2チャネルあります．図5・1の各ユニットは，次のような動作をします．

〔1〕 **タイマデータレジスタ**　データ長が16ビットのレジスタです．タイマ動作の起動がかかると，ϕクロック20個に1回デクリメントします．そして，レジスタの内容が0000Hになると，リロードレジスタの内容がタイマデータレジスタにコピー（リロード）されます．CPUはタイマデータレジスタを8ビット単位で読書きでき，リードする場合，通常まず下位8ビットを読み，続いて上位8ビットを読みます．

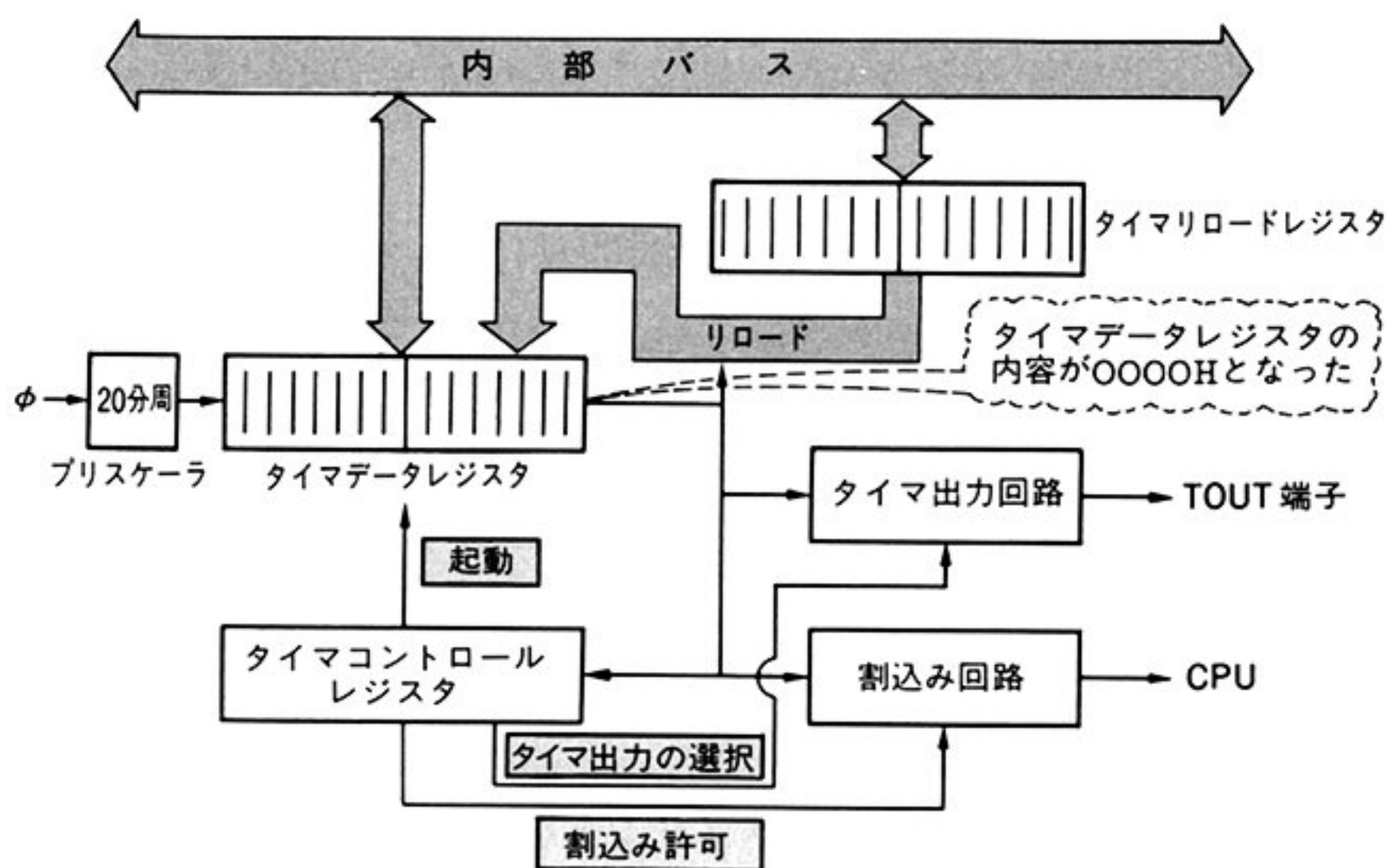



図5・1　プログラマブルリロードタイマのブロック図

〔2〕 **タイマリロードレジスタ**　データ長が16ビットのレジスタです．このレジスタは，タイマデータレジスタにコピーされる定数を保持しておくためのレジスタです．CPUからデータを8ビット単位で読書きできます．

〔3〕 **タイマ出力回路**　TOUT端子状態を制御する回路です．TOUT端子は，アドレスのA_{18}と兼用しています．“1”出力，“0”出力，トグル出力の3つのうちの1つを選択できます．なお，この回路はチャネル0にはなく，チャネ

ル1だけにあります．

〔4〕 **割込み回路**　タイマデータレジスタの内容が0000Hとなると，タイマインタラプトフラグ（TIF）が“1”にセットされます．タイマインタラプトイネーブル（TIE）が“1”に設定されていると，CPUへ割込みを要求します．

〔5〕 **タイマコントロールレジスタ**　タイマ動作を制御するレジスタです．それぞれのビットの機能は**図5・2**に示されています．TOC0，TOC1ビットを除いて，“0”はチャネル0を“1”はチャネル1を表しています．

〔6〕 **プリスケーラ**　入力はϕクロックで，分周比は20です．しかしながらこの分周比は変更することができません．

TOC1	TOC0	A_{18}/TOUTの選択
0	0	A_{18}
0	1	TOUT（トグル）
1	0	TOUT（“0”出力）
1	1	TOUT（“1”出力）

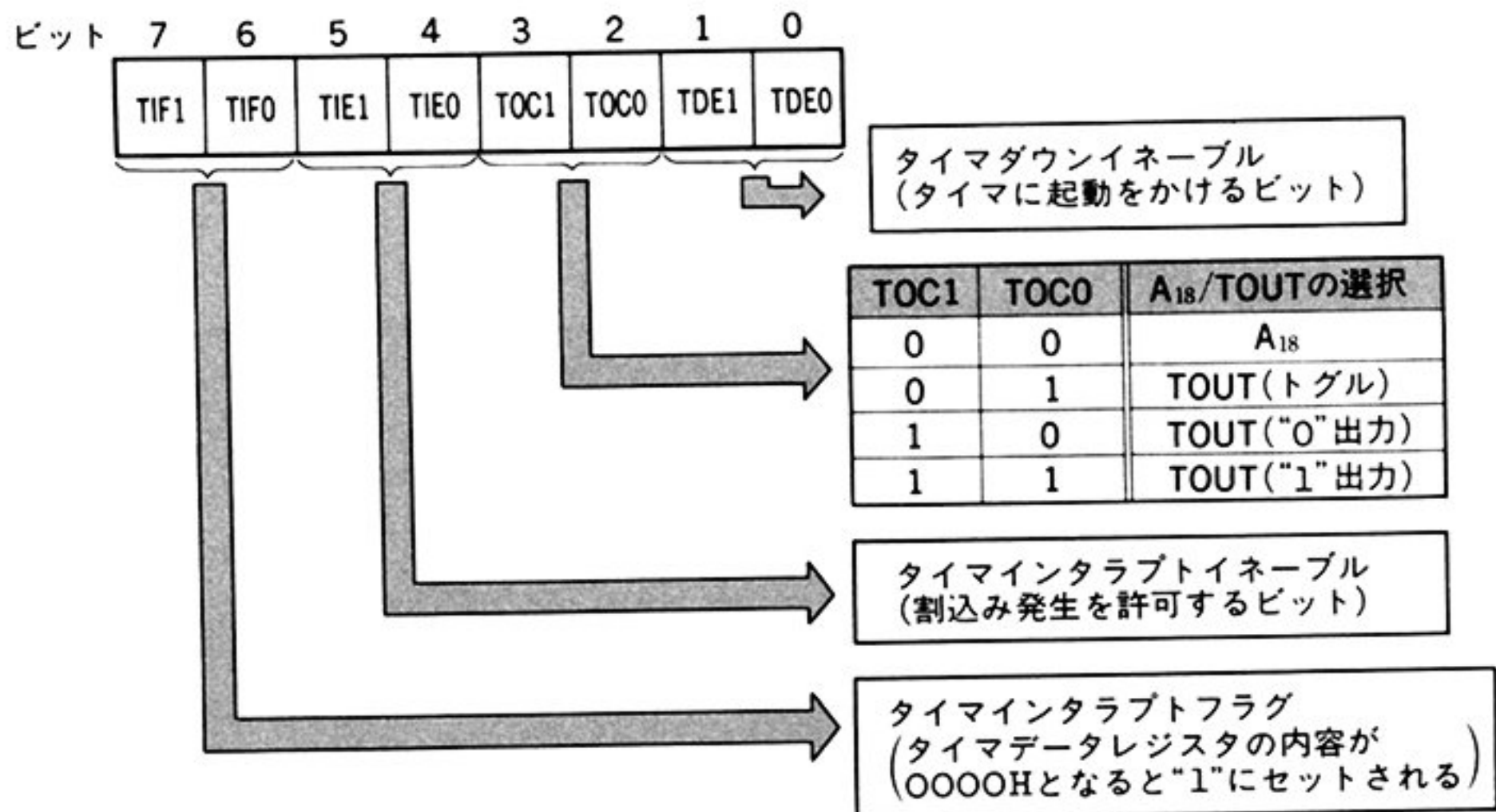



図5・2 タイマコントロールレジスタ

タイマチャネル0の出力端子

タイマチャネル0には，タイマ出力端子がありません．しかし，タイマ出力が必要な場合には，ASCIの$\overline{RTS_0}$端子をタイマ出力端子とみなして使用できます．

$\overline{RTS_0}$端子の状態は，ASCIのコントロールレジスタ内にある$\overline{RTS_0}$ビットの状態が反映されます．したがって，TOUT端子同様に使う場合，タイマ割込みルーチン内のソフトウェアでこのビットに“1”または“0”を書き込めば実現できるでしょう．

5・2 リロードタイマの動作原理

〔1〕 概　要　64180のタイマは，ダウンカウント方式のプログラマブルリロードタイマです．タイマデータレジスタは，20ϕクロックごとに1つずつダウンカウントします．そして，その内容が0000Hとなったとき，次の3つの動作が同時に行われます．①タイマリロードレジスタの内容がタイマデータレジスタへコピー（リロード）されます，②タイマ出力端子（TOUT）のレベルが変化します，③TIF（タイマインタラプトフラグ）を“1”にセットして，CPUに割込みを要求します．その後，再びリロードされた値からダウンカウントします．

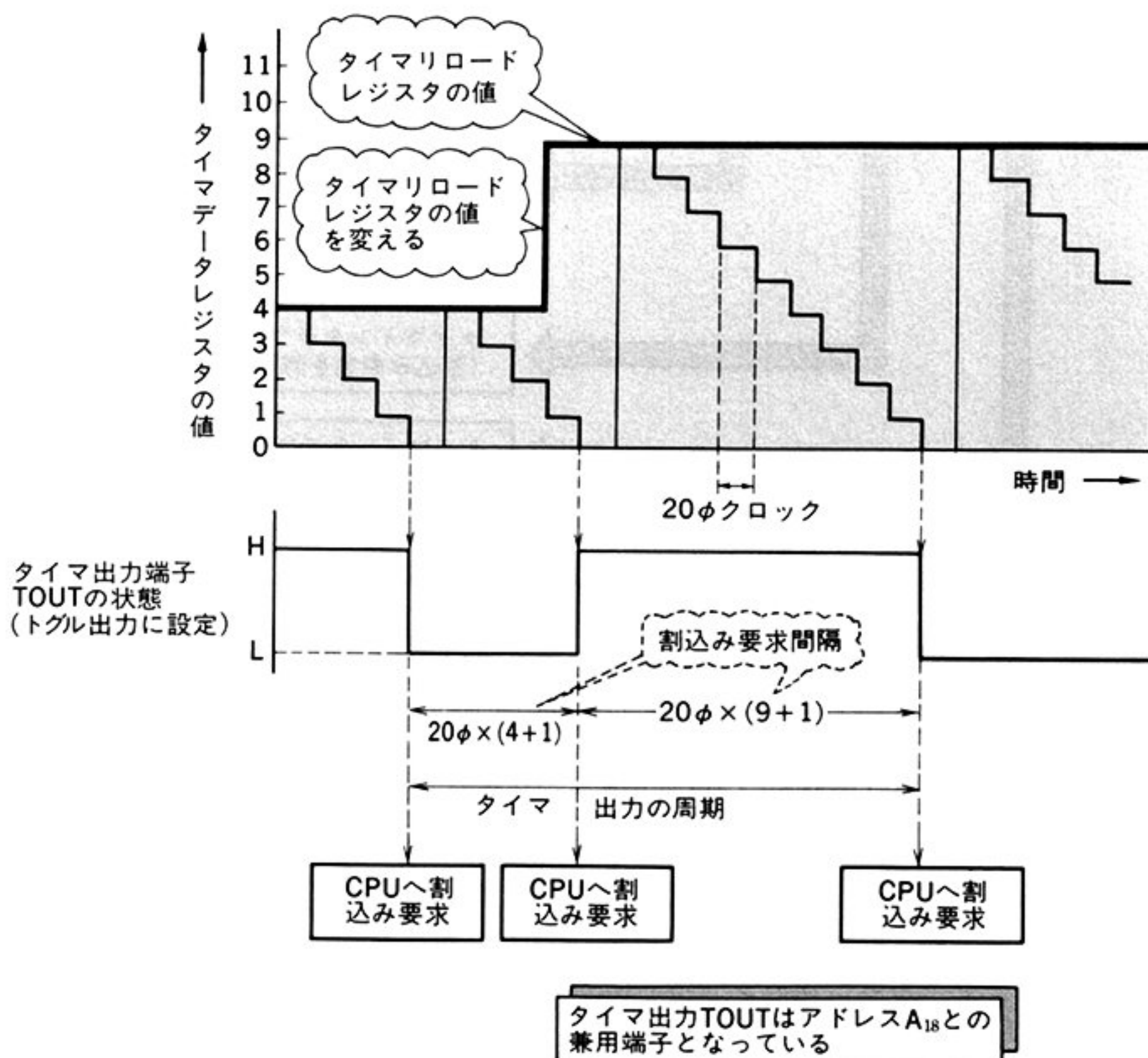



図 5・3　リロードタイマの動作原理

〔2〕 **タイマ出力**　トグル出力を選択している場合，**図5・3**のように前の状態を反転します．タイマリロードレジスタの値を変えなければ，デューティ比50％の矩形波が得られます．図では，途中でタイマリロードレジスタの値を変えているので，タイマ出力の周期が変化しています．

〔3〕 **タイマ割込み**　アップカウント方式のリロードタイマと異なり，タイマリロードレジスタに値を設定した後の最初のリロード時からこの設定値が適用されます．この際，割込み周期は，タイマリロードレジスタの値に1を加えて20 tcycを乗じた時間となります．ここでtcycは動作周波数の逆数です．TIFはタイマコントロールレジスタをリードした後，続けてタイマデータレジスタの上位または下位8ビットをリードすることによりリセットします．通常は，タイマ割込みルーチンの中でリセットします．

〔4〕 **タイマ出力の周期と割込み要求の周期**　それぞれの最小値は，80 tcycと40 tcycです．そのときのタイマリロードレジスタの値は0001Hです．逆に最大値は，0000Hとした場合です．そのときのタイマ出力の周期は65537（10000H＋1）×40 tcycで，割込み要求の周期は65537（10000H＋1）×20 tcycです．

5・3 タイマの使用例

〔1〕**音　出　力**　タイマチャネル1を使用し，そのタイマ出力端子TOUTよりパルスを出力して，音階を出す場合の例を示します．

TOUT出力をトグル出力にした場合，タイマデータレジスタの内容が0000Hとなるごとに前の状態を反転するので，その反転する時間を調節することによりある一定周期の矩形波を出力します．そして，この矩形波をアンプを通じてスピーカに入力すれば，音として取り出すことができます（**図5・4**参照）．

今，タイマリロードレジスタ1（RLDR1）の値を途中で変えなければ，その音の周期は**図5・5**の式で表されます．この式から，動作周波数とRLDR1の値を与えると，任意の周期の音が出力されます．逆に，希望する音を出すためには，RLDR1の値をこの式から逆算して求めればよいでしょう．

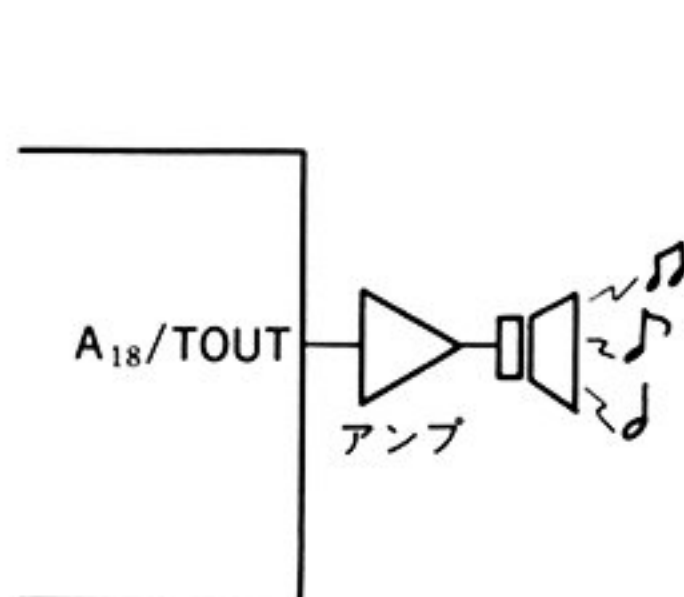
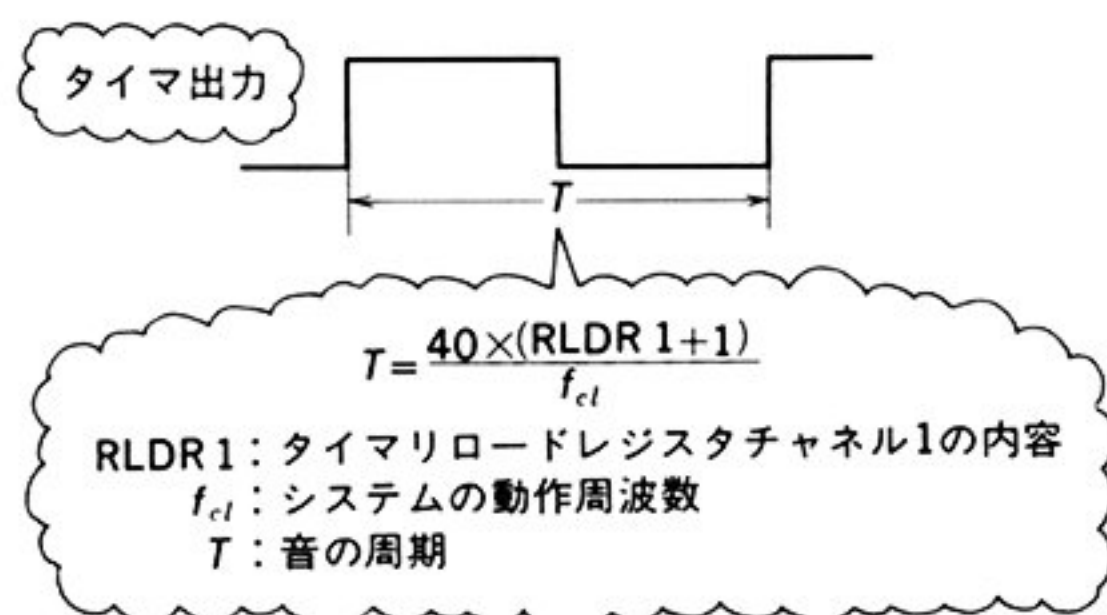



図 5・4　音出力　　　　図 5・5　音の周期

表 5・1　音出力データ

音階		RLDR1 の値	
記号	周波数〔Hz〕	10 進	16 進
ド	523.25	293	0125
シ	493.88	310	0136
ラ	440.00	348	015C
ソ	392.00	391	0187
ファ	349.23	439	01B7
ミ	329.63	465	01D1
レ	293.66	522	020A
ド	261.63	586	024A

f_{cl}=6.144MHz 時

この方法により，7音階を与えるRLDR1の値を求めたものが**表5・1**です．ただし，動作周波数f_{cl}は6.144 MHzです．例として，「ラ」の音を出す場合のプログラミング例を**図5・6**に示しました．このリロードタイマを使用すれば，簡単なプログラムで任意の音を出せることがわかると思います．

```
LD A,5CH              ┐
OUTO (TMDR1L), A      │
OUTO (RLDR1L), A      │ タイマデータレジスタとリロードレジスタに「ラ」の音データを与える
LD A, 01H             │
OUTO (TMDR1H), A      │
OUTO (RLDR1H), A      ┘
LD A, 00000110B       ┐ タイマ出力をトグルに選択し，タイマチャネル1のイネーブルビットを
OUTO  (TCR), A        ┘ "1"に設定して，タイマチャネル1を起動させる
```

図5・6 「ラ」の音を出すプログラム例

〔2〕 **タイムベース**　周期0.1秒のソフトタイマを考えます．割込みを使用し，CPUへ0.1秒ごとに割込み要求を出す場合です．割込み要求はタイマデータレジスタが0000Hとなったとき発生するので，その周期はT/2で与えられます．したがって，0.1秒ごとに割込み要求を発生させるために，タイマリロードレジスタへは40 000−1=39 999（16進では，9C3FH）を与えます．ただし，**図5・5**の式を使用し$T=0.2$秒，$f_{cl}=8$MHzとしています．この値をRLDRに書き込み，割込みをイネーブルにした後タイマを起動させます．

6. 非同期方式シリアルI/O

非同期方式シリアルI/Oは，パソコン，プリンタなど多くのアプリケーションで使用されている通信の代表的な方式です．ACIAやUARTなどいろいろな呼び方がありますが，64180ではASCIと呼んでいます．ASCIは，スタートビットとストップビットをデータストリングの中にもっているのが特徴であり，スタートビットで同期をとるためクロック信号を送る必要がありません．

6・1 シリアルコミュニケーションインタフェースの分類

マイクロコンピュータに内蔵されているシリアルコミュニケーションインタフェースは，**図6・1**に示すように非同期方式シリアルI/Oと同期方式シリアルI/Oに分類することができます．

〔1〕 **非同期方式シリアルI/O**　別名「調歩同期式シリアルI/O」と呼ばれる非同期式シリアルI/Oは，パソコン，プリンタなど多くのアプリケーションで使用されています．64180ファミリーでは，この非同期方式シリアルI/OをASCI（Asynchronous Serial Communication Interface）と呼んでいます．

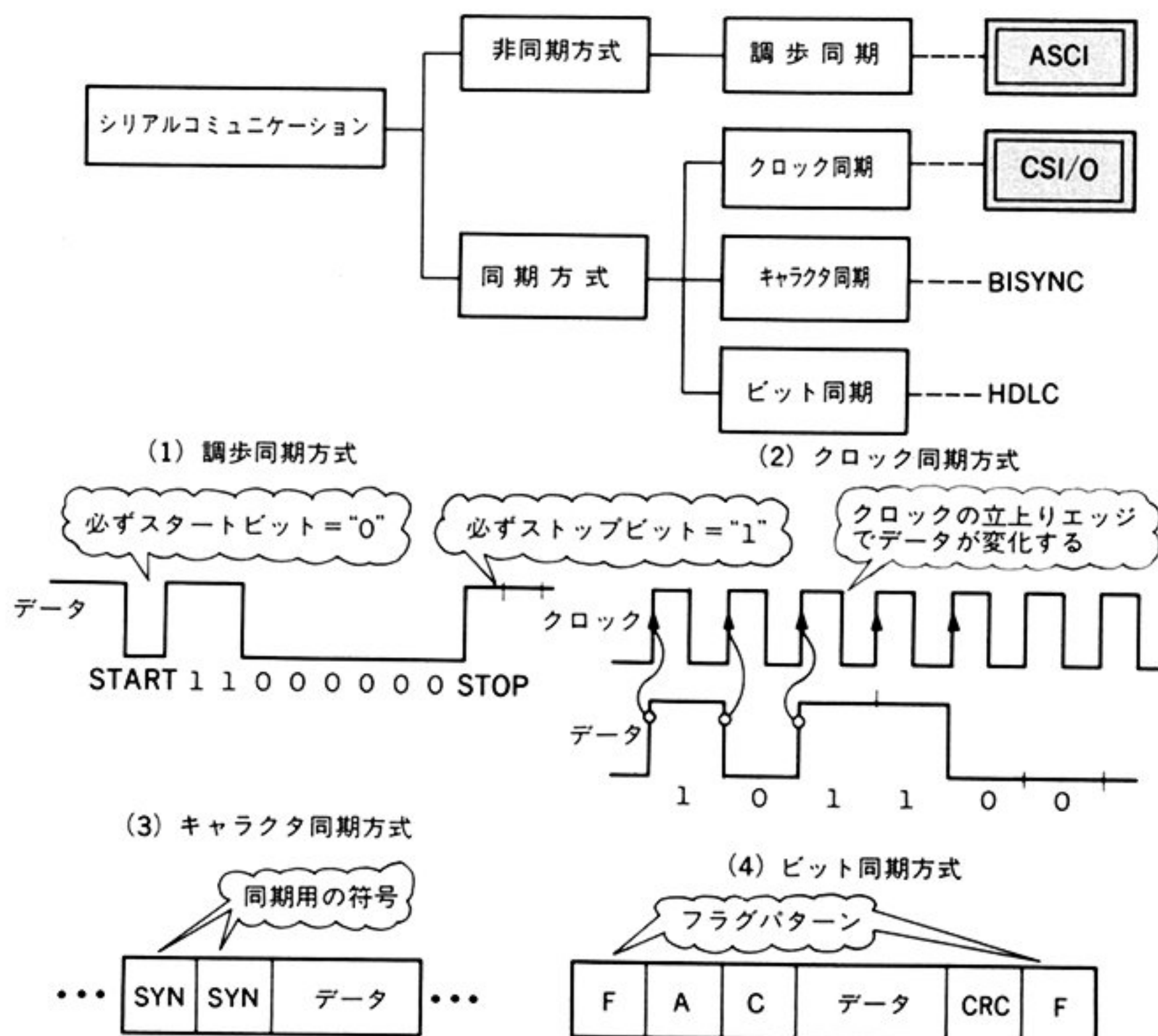



図6・1 シリアルコミュニケーションインタフェースの分類

ASCIの大きな特徴は，送信側のクロックタイミングと受信側のクロックタイミングが非同期でデータの送受信ができることです．このため，データストリングの中に同期をとるためのスタートビットおよびストップビットが必要です．

〔2〕 **同期方式シリアルI/O** 同期方式シリアルI/Oは、①クロック同期方式，②キャラクタ同期方式，③ビット同期方式に分類できます．このうち64180ファミリーでは，クロック同期方式シリアルI/O(**CSI/O**：Clocked Serial Input Output）を内蔵しています．

CSI/Oは，マイコンとマイコンのインタフェース，マイコンと周辺LSIとのインタフェースなどに使用されます．CSI/Oの特徴は送受信のビットごとの同期を，クロックの立上りエッジまたは立下りエッジでとることです．このため，送信側と受信側が同じタイミングのクロックを用いてデータのやりとりをすることが必要となります．

キャラクタ同期方式シリアルI/Oは，同期をとるための特定符号（SYN符号）をデータの前につけて伝送する方式です．SYN符号を2つつけてデータを送る方式をBISYNCと呼びます．SYN符号を受信すると，その次からデータが連続して送られてくるものとして文字を組み立てていきます．

ビット同期方式シリアルI/Oは，伝送路に一定のビットパターン（フラグパターン）を送出しておくことにより，送信側と受信側の同期をとる方式です．フラグパターン以外のデータを受信するとデータが送られてきたと判断し，再びフラグパターンを検出するまでその間の信号をデータとみなします．

6・2 ASCIの構成と機能

64180ファミリーは，ASCIを2チャネル内蔵しています．ASCIは**図6・2**に示すように，受信側がレシーブシフトレジスタ，レシーブデータレジスタから成り，送信側がトランスミットシフトレジスタ，トランスミットデータレジスタから成っています．さらに，送信/受信の制御のためのコントロールレジスタ2個と，ASCIの状態を示すステータスレジスタから構成されています．

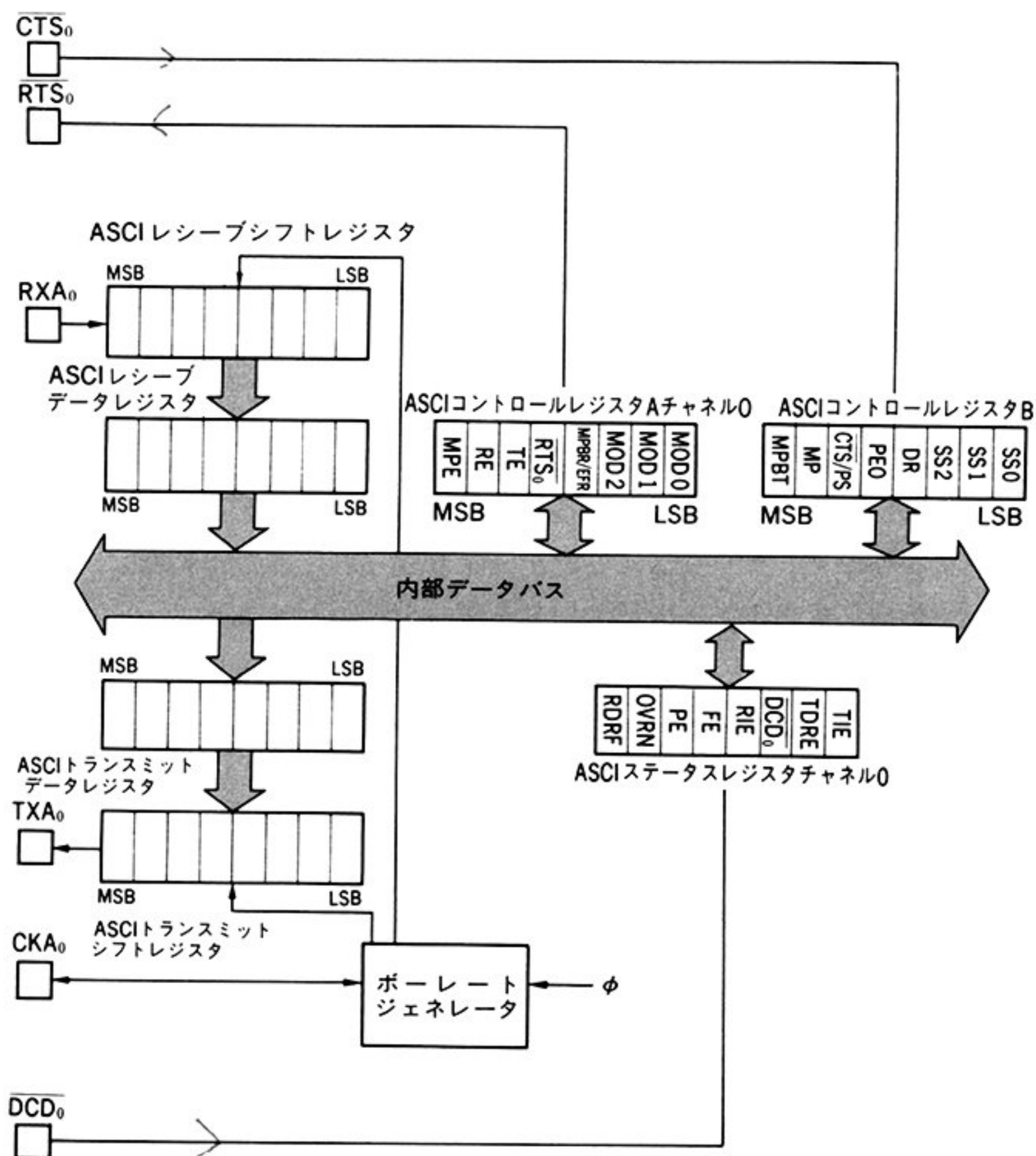



図 6・2 ASCI ブロック図

〔1〕 **ASCIによるデータ受信**　コントロールレジスタにある**RE**(Receive Enable)＝"1"で受信可能な状態となります．RXA_0端子から入力されたデータはまず，レシーブシフトレジスタへ1ビットずつ取り込まれシフトされます．1バイトのデータの受信が終了すると，受信したデータは自動的にレシーブデータレジスタへ転送されます．この状態で次の新しい受信データが，再びレシーブシフトレジスタへ取込み可能となります．つまり，受信部はレシーブシフトレジスタとレシーブデータレジスタの2重バッファ構成となっていますので，1バイト受信後，次のデータの受信が完了する前までに受信データをレシーブデータレジスタから読めばよいようになっています．レシーブデータレジスタに受信データが送られてくると，ステータスレジスタの**RDRF**(Receive Data Register Full)が"1"にセットされます．

〔2〕 **ASCIによるデータ送信**　コントロールレジスタの**TE**(Transmit Enable)＝"1"で送信可能な状態となります．1バイトの送信データは，まずトランスミットデータレジスタへ書き込まれます．トランスミットシフトレジスタがからになると，トランスミットデータレジスタから自動的にトランスミットシフトレジスタへ転送され，1ビットずつTXA_0端子からデータが送信されます．送信側もトランスミットデータレジスタとトランスミットシフトレジスタの2重バッファ構成になっていますので，トランスミットデータレジスタがからになる(ステータスレジスタにある**TDRE**：Transmit Data Register Emptyが"1")と次の送信データを書き込むことができます．つまり，データをTXA_0端子から送信しながら次のデータも書くという並列処理ができますので，複数バイトのデータを高速に転送できます．

6・3 ASCI のデータフォーマットおよび転送速度

〔1〕 **データフォーマット**　図6•3に示すようにASCIコントロールレジスタのMOD0～MOD2を用いることにより，送受信のデータフォーマットを選択できます．データフォーマットは①スタートビット，②データ，③パリティビット，④ストップビットから構成されます．スタートビットはデータの最初のビット位置を示し必ず"0"です．一方，ストップビットはデータの最後を示し必ず"1"です．ストップビットは1ビットまたは2ビットのいずれかを選択できます．スタートビットに続くデータは7ビットと8ビットがあり，用途によりユーザが使い分けることができます．たとえば，ASCIIコードを送受信するときは7ビットのデータを使用します．

MOD2	MOD1	MOD0				
0	0	0	START	7bit data		STOP
0	0	1	START	7bit data		2STOP
0	1	0	START	7bit data	PARITY	STOP
0	1	1	START	7bit data	PARITY	2STOP
1	0	0	START	8bit data		STOP
1	0	1	START	8bit data		2STOP
1	1	0	START	8bit data	PARITY	STOP
1	1	1	START	8bit data	PARITY	2STOP

図6・3　データフォーマットの種類

さらに，データ送受信の際，伝送路におけるエラーを検出するためにパリティビットの付加もできます．偶数パリティか奇数パリティかをコントロールレジスタの **PEO**(Parity Even or Odd)ビットで選択できます．このように，8種類のデータフォーマットが選択できます．

〔2〕 転送速度　転送速度(一般にボーレート)を発生する方法は，**図6・4**に示すように内部クロックを使用する場合と外部クロックを使用する場合の2通りがあります．

内部クロックからボーレートを発生する場合，内部クロック ϕ はボーレート選択器，プリスケーラおよびサンプリングレート選択器を通って任意のボーレートとなります．

一方，外部クロックからボーレートを発生する場合，外部クロック端子CKAより入力されたクロックは，プリスケーラとボーレート選択器を通らず直接サンプリングレート選択器の入力となり，ボーレートとなります．

サンプリングレート選択器は，ボーレートの16倍または64倍のサンプリングクロックを生成するために必要であり，この指定はコントロールレジスタBのDR(Divide Ratio)ビットで行います．また，内部クロックからボーレートを生成する場合に限り，プリスケーラ(÷10または÷30の選択)およびボーレート選択器(÷1～÷64)が有効となります．

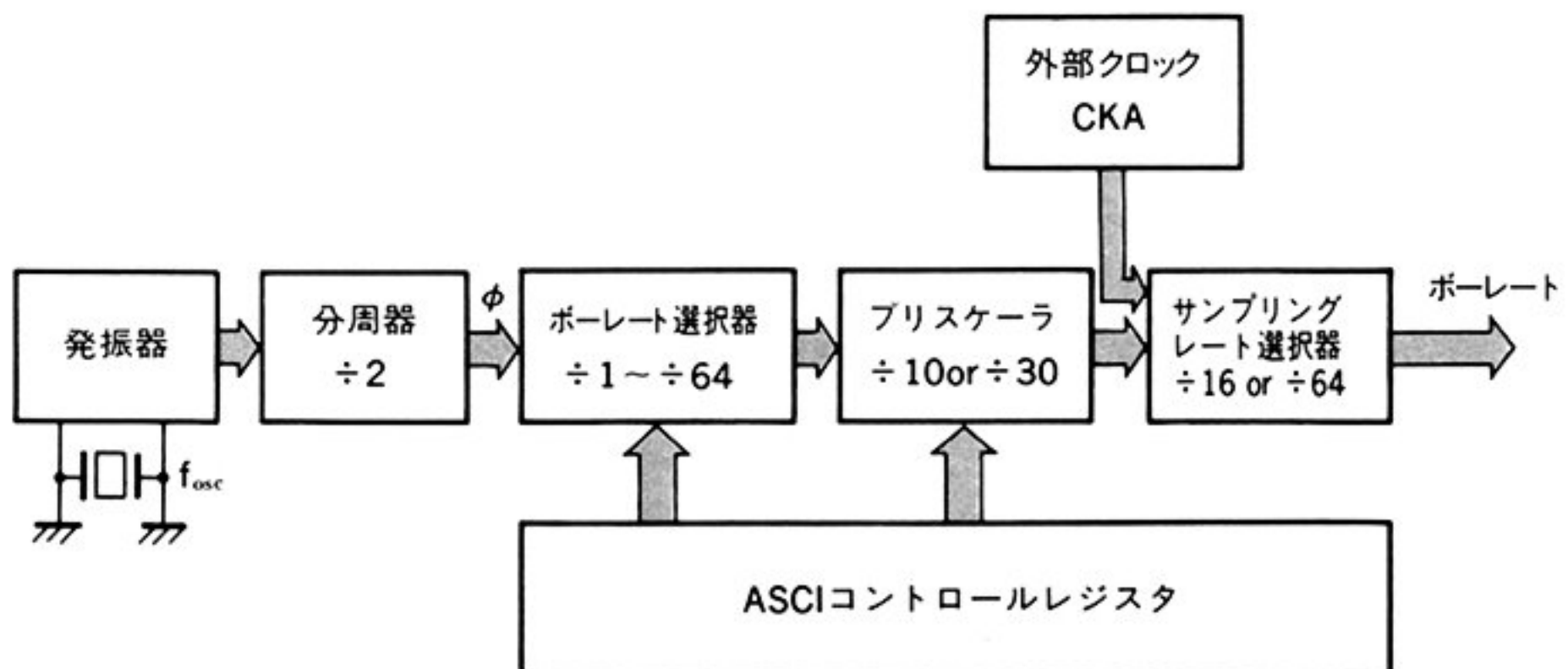



図6・4 ASCIのクロック発生回路

6・4 ASCIのエラーチェック機能

ASCIが備えているエラーチェック機能には，①フレミングエラー，②パリティエラー，③オーバランエラーの3種類があります．**図6•5**にエラーの種類と概要を示します．

〔1〕 **フレミングエラー**　ASCIのデータフォーマットでは，データの最後

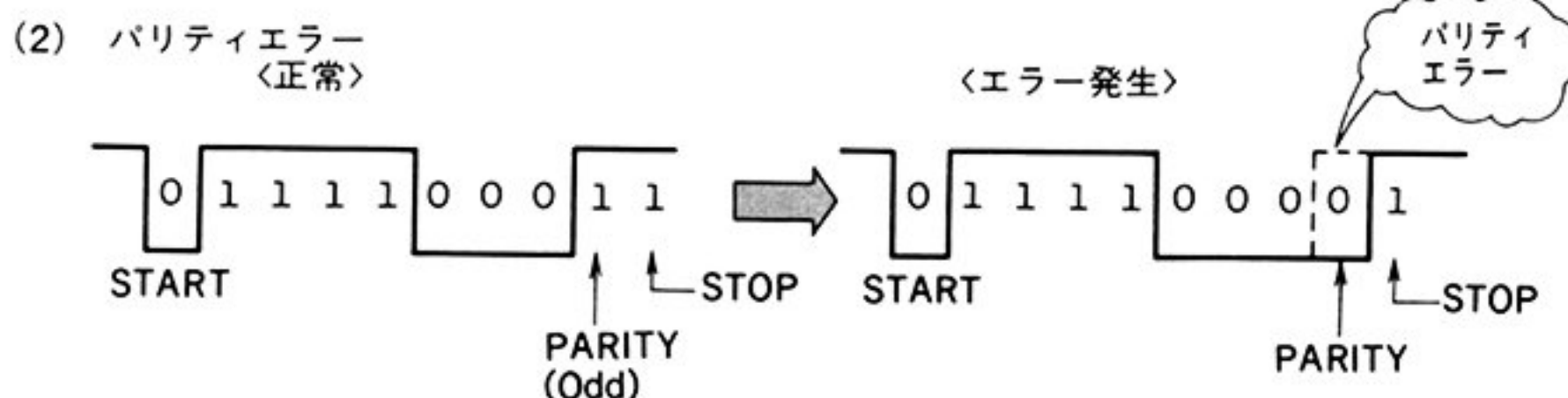
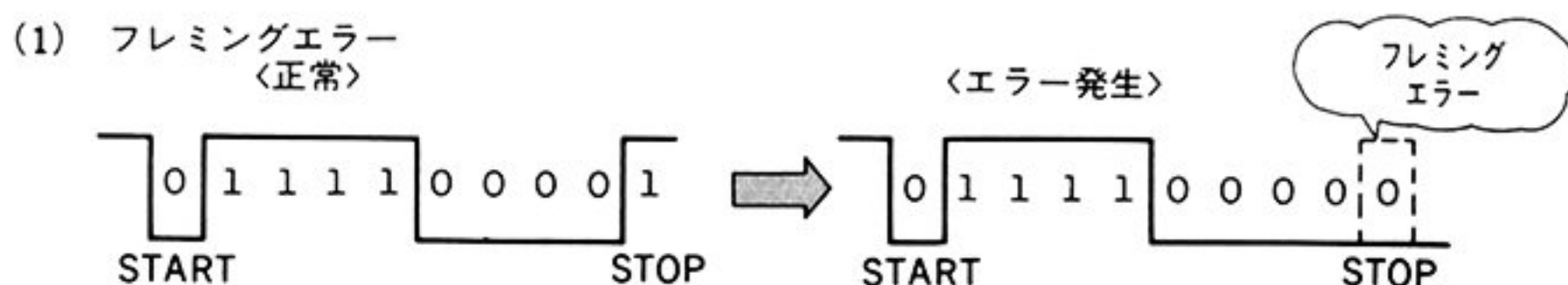
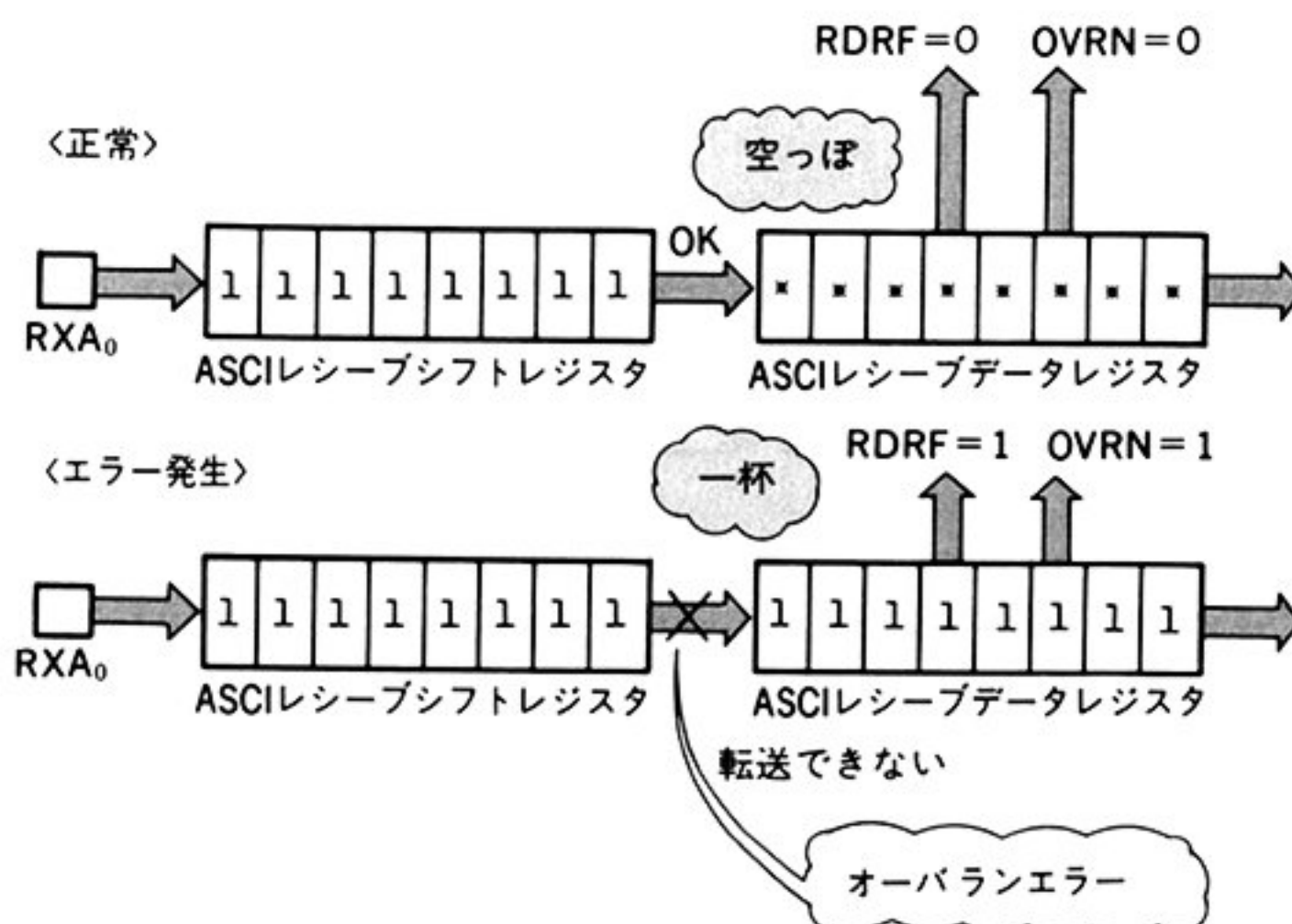



図 6・5　エラーの種類

は必ずストップビット（“1”レベル）でなければなりません．しかし，伝送路を通る間にストップビットが“1”から“0”に化けることがあります．このように変化したデータを受信すると，フレミングエラーが生じたとして，ASCIステータスレジスタのFE（Framing Error）フラグを“1”にセットします．

〔2〕 **パリティエラー**　送信データが伝送路を通り正しく受信されたかチェックするために，パリティエラーチェック機能を用います．データの最後にパリティビットをつけて，データとパリティビットの“1”の数が偶数になるようにするのを偶数パリティ，“1”の数が奇数になるようにするのを奇数パリティと呼びます．送信側と受信側であらかじめ偶数パリティか奇数パリティのどちらで交信するのかを決めておくと，受信データとパリティビットの“1”の数でエラー検出ができます．パリティエラーを検出するとASCIステータスレジスタのPE（Parity Error）フラグに“1”をセットします．

〔3〕 **オーバランエラー**　ASCIの受信側は，ASCIレシーブシフトレジスタとASCIレシーブデータレジスタの2重バッファ構成ですが，受信したデータがASCIレシーブデータレジスタにまだ入っていて，CPUからまだ読まれていないときに次のデータの受信が完了した場合に，オーバランエラーが発生します．RDRF＝“1”の状態で次の受信データの最後のビットまで受信したとき，ASCIステータスレジスタのOVRN（OVer RuN error）フラグを“1”にセットします．

これら4種類のエラーが発生すると，エラーフラグをセットするだけでなく割込みも発生させることができます．CPUはエラーチェックの結果に基づいて，それぞれのエラー処理を行い正しい送受信データのみ受け取ることが可能です．

6・5 モデム制御信号

モデム制御信号として，チャネル0には$\overline{\mathrm{CTS}_0}$（クリア・ツー・センド），$\overline{\mathrm{RTS}_0}$（リクエスト・ツー・センド），$\overline{\mathrm{DCD}_0}$（データ・キャリア・ディテクト）の3端子があり，チャネル1には$\overline{\mathrm{CTS}_1}$端子があります．これらのモデム制御信号は**図6・6**に示すように，モデムと接続することにより多くのシリアル通信を必要とするアプリケーションに使用できます．$\overline{\mathrm{CTS}_0}$および$\overline{\mathrm{RTS}_0}$端子は，64180からモデムへ送信するときの制御信号として使用され，$\overline{\mathrm{DCD}_0}$端子は64180がモデムから受信するときの制御信号として使用されます．モデム制御信号を用いた送信および受信動作を，**図6・7**を用いて説明します．

〔1〕**送信動作**　$\overline{\mathrm{CTS}_0}$端子はTDRE（トランスミットデータレジスタエンプティ）フラグを読出し可能とし，かつ送信データをモデムへ送るときの制御信号として使用されます．$\overline{\mathrm{CTS}_0}$＝“1”の間は，TDREは強制的に“0”レベルとなりますが，$\overline{\mathrm{CTS}_0}$＝“0”になるとTDREはTDR$_0$（ASCIトランスミットデータレジスタ）の状態に従って変化します．つまり，$\overline{\mathrm{CTS}_0}$＝“0”，TDRE＝“1”

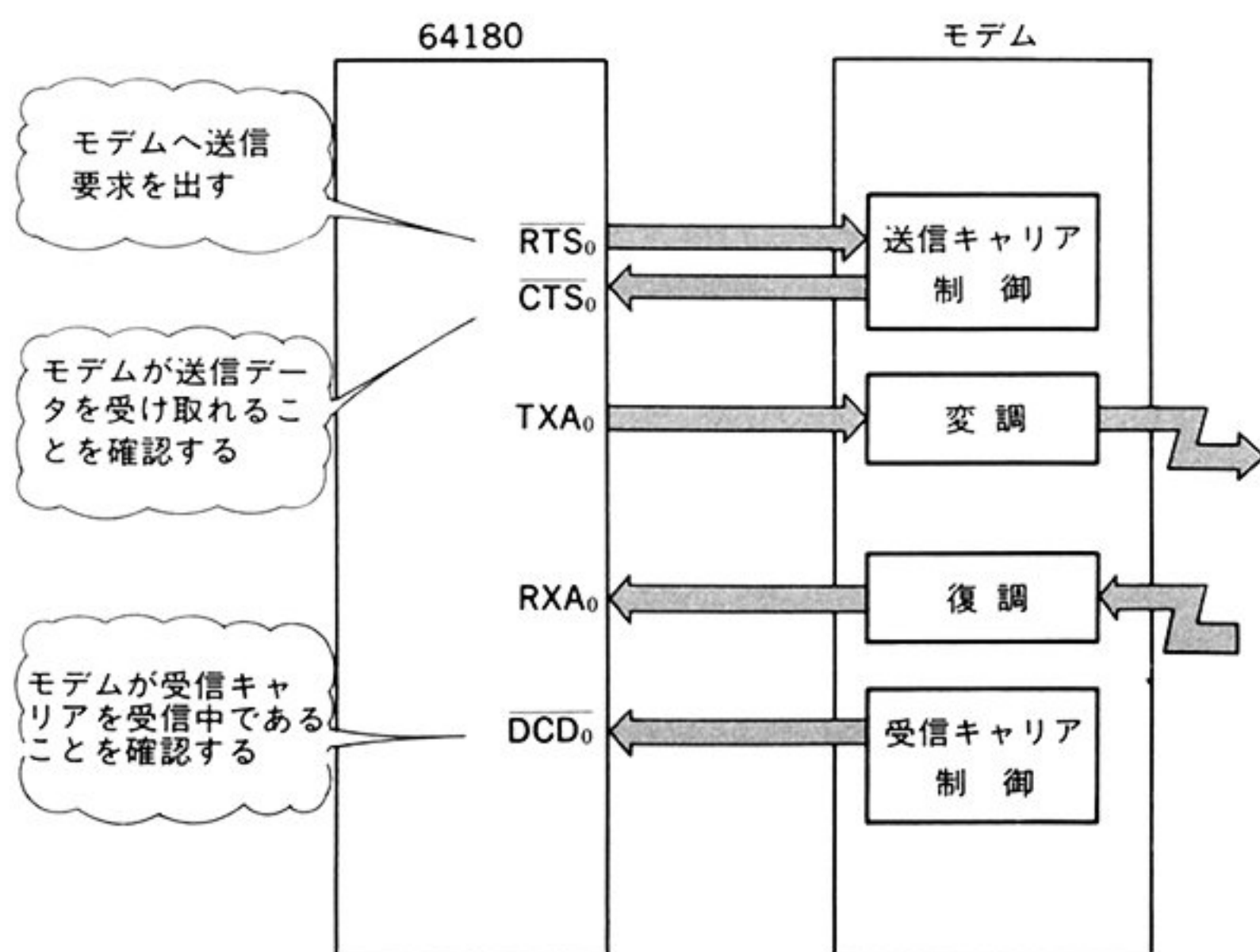



図6・6　モデムとの接続

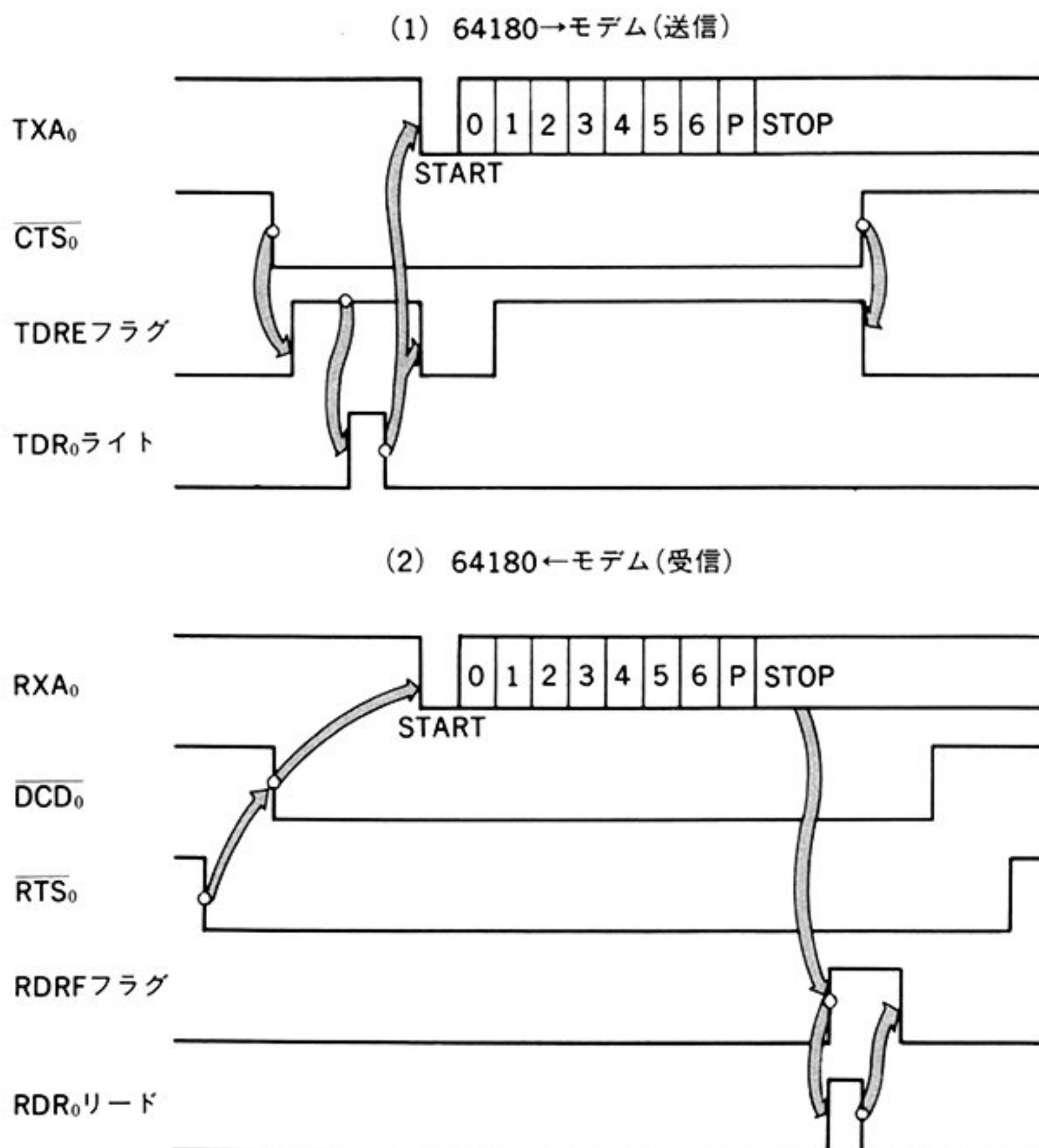



図6・7 送/受信動作タイミング

のときに TDR_0 へ送信したいデータをライトすると，TXA_0 端子よりデータが送信されます．送信が終了すると $\overline{CTS_0}$ 端子を“1”レベルにして処理を終わります．

〔2〕 **受信動作**　64180はモデムに対する送信要求のため，$\overline{RTS_0}$ 端子を“0”レベルにクリアします．モデムから送信するデータがある場合は，$\overline{DCD_0}$ 端子が“0”レベルとなり，RXA_0 端子よりデータが送られてきます．$\overline{DCD_0}$ 端子はモデムのキャリア検出を示す信号であり，“0”レベルの間は受信動作が可能です．“1”レベルになると受信動作はただちに中止されます．RDRF(レシーブデータレジスタフル)フラグが“1”になるのを確認したら，RDR_0 (ASCI レシーブデータレジスタ）をリードし，$\overline{DCD_0}$＝“1”となり受信処理が終了すると $\overline{RTS_0}$ を“1”へセットします．

6・6 ASCI の使用例

ASCI の具体的応用例の中には，64180↔モデムのインタフェース，64180↔ACIA（HD6850/HD6350）のインタフェース，64180↔UART（8251）のインタフェースが考えられます．ここでは，64180↔ACIA（HD6350）のインタフェースをハードウェア，ソフトウェアの両面から説明します．

〔1〕 **64180↔ACIA（HD6350）とのインタフェース**　64180でASCIと呼んでいる非同期シリアルコミュニケーションと同等機能の周辺LSIがACIA（Asynchronous Communication Interface Adapter）HD6350です．64180とACIAは図6•8に示すように，RXA_0（64180）←TxData（HD6350），TXA_0→RxData，$\overline{RTS_0}$→$\overline{CTS}$，$\overline{CTS_0}$←$\overline{RTS}$と送受信に関する信号を互いにクロスして接続します．$\overline{DCD}$端子は64180もHD6350も内蔵していますが，モデムとの接続ではないのでともに"0"レベルに固定します．HD6350はRxCLKとTxCLKが別々になっていますが，64180は1本のクロックCKA_0しかありませんので，送信クロックTxCLKと受信クロックRxCLKは同一の転送速度とし，CKA_0→RxCLK/TxCLKと接続します．

〔2〕 **ソフトウェア例**　図6•9にACIAとのインタフェースで必要な初期設

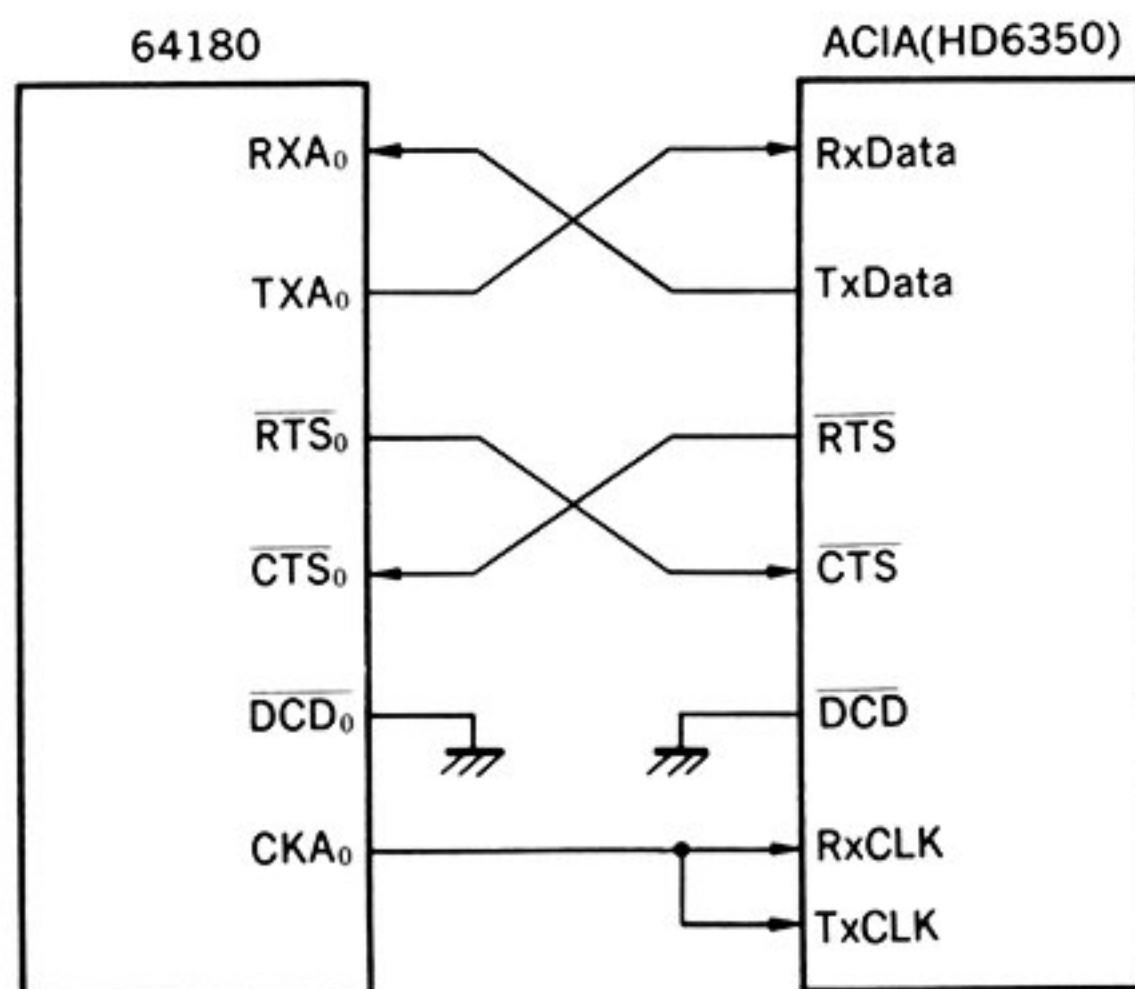



図6・8　64180とACIAとのインタフェース

初期設定

```
    LD    A,02H          } ボーレート1200BPSに設定
    OUT0  (CNTLB0),A     }   (at φ=6.144MHz)
    LD    A,64H          } 送受信許可(TE=1, RE=1)
    OUT0  (CNTLA0),A     } 8ビットデータ+1ストップビット
    LD    A,08H          } 受信割込み許可(RIE=1)
    OUT0  (STAT0),A      } 送信割込み禁止(TIE=0)
    EI                   } 割込み許可(IEF=1)
```

送信動作

```
T1:IN0  A,(STAT0)        }
    BIT  1,A             } TDRE="1"かチェック
    JR   Z,T1            }
    LD   A,XXH           } 送信データ XXH をTDR0
    OUT0(TDR0),A         } へセット
```

受信動作

```
S1:IN0  A,(STAT0)        }
    BIT  7,A             } RDRF="1"かチェック
    JR   Z,ERROR         }
    AND  70H             } フレミングエラー, オーバランエラー,
    JR   NZ,ERROR        } パリティエラーのチェック
    IN0  A,(RDR0)          受信データリード
         ⋮
ERROR: [エラー処理]
         ⋮
    RET
```

図 6・9 ソフトウェア例

定，送信動作，受信動作のソフトウェア例を示します．ASCIの初期設定では，ボーレートをCNTLB0（ASCIコントロールレジスタB）で設定し，送受信許可フラグTE/REおよびデータフォーマットをCNTLA0（ASCIコントロールレジスタA_0）で設定し，RIE="1"，TIE="0"をSTAT0（ASCIステータスレジスタ0）で設定後，EI命令で割込み許可の状態にします．一方，送信動作ではTDRE="1"か否かをチェック後，送信したいデータをTDR0へライトします．また，受

信動作ではRDRF＝“1”か否かをチェックした後エラーチェックをし，受信データをRDR0（レシーブデータレジスタ0）からリードします．ソフトウェアの基本的アルゴリズムは同じですので，ここに示すソフトウェア例は他の周辺LSIと接続するときでも使用できます．

ASCIのストップビット長

ASCIのストップビット長を1ビットにするか2ビットにするかにより，送受信がうまくいくときとうまくいかないときがあります．たとえば，64180のASCIが受信側で，送信側にストップビット長が1ビットと2ビットの2種類ACIAがあったとします．受信側のストップビットを2ビットで選択した場合，送信側のストップビットが2ビットだと正常に受信しますが，送信側のストップビットが1ビットだとフレミングエラーとなります．一方，受信側のストップビットを1ビットで選択した場合，送信側のストップビットが1ビットでも2ビットでも正常に受信します．したがって，受信側では1ビット，送信側では2ビットのストップビットを選択するのが，システム設計のノウハウです!?

7. クロック同期方式シリアルI/O

高速シリアル通信を実現するのがクロック同期方式シリアルI/Oであり，マイコン間通信やマイコンと周辺LSI間通信に適しています．クロックや立下りエッジに同期してデータの送受信を行うため，短い距離のシリアル通信として用いられます．内部のトランスミット/レシーブデータレジスタが1個で送信と受信で共用しているため，半2重通信方式をサポートしています．

7・1 CSI/Oの構成と機能

クロック同期方式シリアルI/Oポート（CSI/O）は、非同期方式シリアルI/O（ASCI）に比べ，転送速度が最大400 kbps（ϕ＝8 MHz）と非常に高速です．このため，伝送距離の短い機器内通信で採用される場合が多く，マイコン↔マイコン，マイコン↔周辺LSI間のデータ転送では重要な役割を果たしています．

64180ファミリーに内蔵しているCSI/Oは図7・1に示すように，RXS，TXS，CKS端子と，CSI/Oトランスミット/レシーブデータレジスタ，およびCSI/Oコントロールレジスタから構成されています．

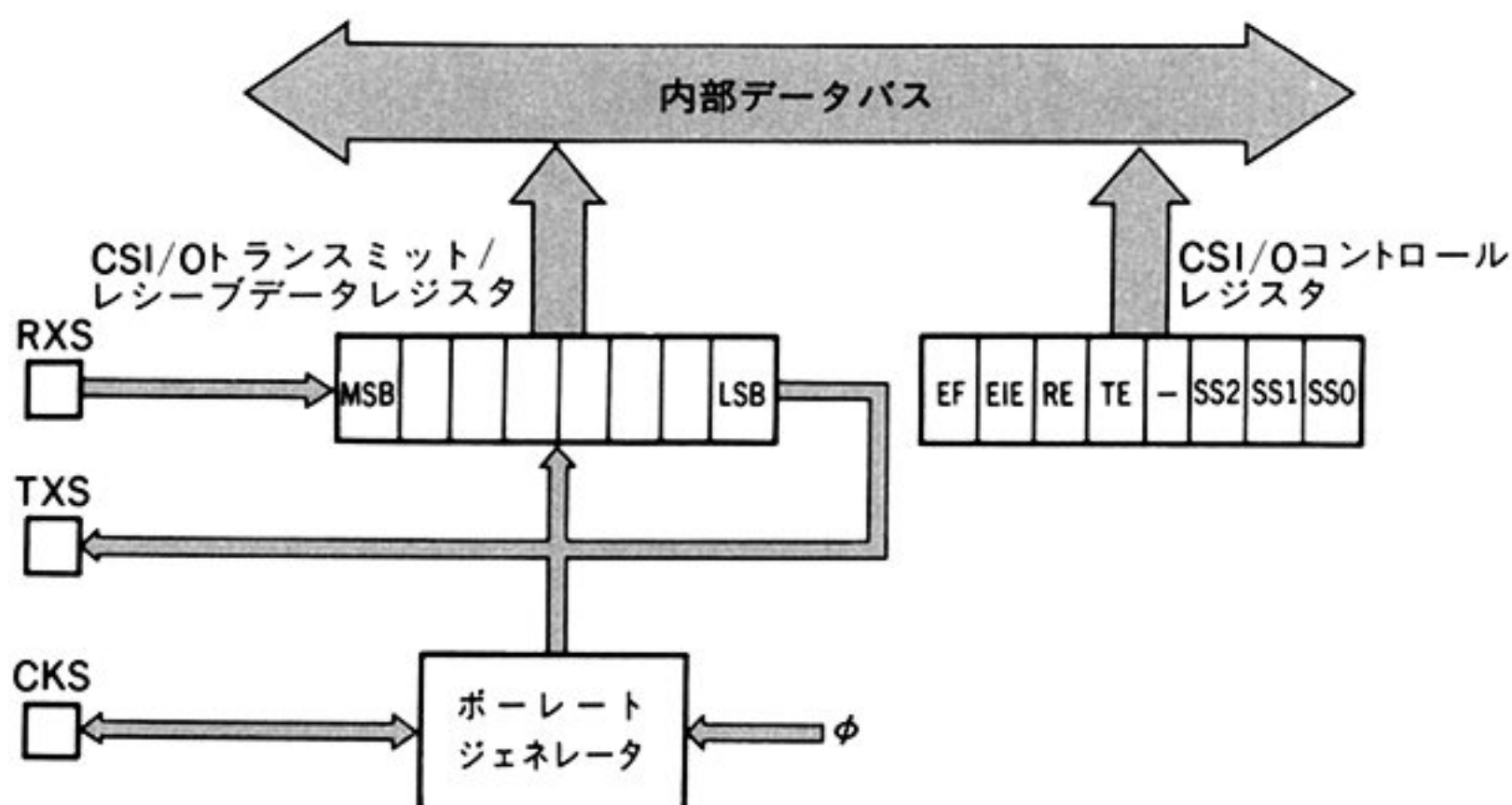



図7・1 CSI/Oブロック図

〔1〕 **CSI/Oトランスミット/レシーブデータレジスタ**　RXS端子（受信端子）あるいはTXS端子（送信端子）を経由して入出力するデータが，CSI/Oトランスミット/レシーブデータレジスタへ入ります．レジスタは1本ですので，同時に送信，受信は行えません．つまり半2重通信方式をサポートしています．また，バッファレジスタをもっていないため，送信途中でこのレジスタにライトすると送信データは変化します．同様に，受信途中でこのレジスタにデータをライトしても受信データは破壊しますので，アプリケーションソフトウェアで制御しなければなりません．

〔2〕 **CSI/Oコントロールレジスタ**　CSI/Oコントロールレジスタの内容は，ボーレートの選択ビット（SS0～SS2），トランスミットイネーブルフラグ

表7・1 ボーレートの設定

CSI/O コントロールレジスタ			ボーレート
SS2	SS1	SS0	
0	0	0	$\phi \div 20$
0	0	1	$\phi \div 40$
0	1	0	$\phi \div 80$
0	1	1	$\phi \div 160$
1	0	0	$\phi \div 320$
1	0	1	$\phi \div 640$
1	1	0	$\phi \div 1\,280$
1	1	1	外部クロック入力

(TE)，レシーブイネーブルフラグ(RE)，エンドフラグ(EF)およびエンドインタラプトイネーブル（EIE）です．

ボーレートは**表7・1**に示すように，SS0～SS2（Speed Select）ビットで外部または内部7種類（$\phi \div 20$，$\div 40$，$\div 80$，$\div 160$，$\div 320$，$\div 640$，$\div 1\,280$）から選択できますので，$\phi = 8$ MHz だと最高 400 kbps となります．

送信の開始はトランスミットイネーブルフラグで制御し，受信開始はレシーブイネーブルフラグで制御できます．また，送受信が完了したことを示すエンドフラグがあり，エンドフラグが"1"のときCPUに割込み要求を許可するエンドインタラプトイネーブルフラグがありますので，シリアル通信を用途に合わせてソフトウェアで制御できます．

7・2 CSI/Oの動作原理

CSI/Oの送受信の動作原理を図7・2および図7・3を用いて説明します.

〔1〕 **送信動作** 送信動作を始めるときは, TE(トランスミットイネーブル) フラグが"0"であるかを確認します. TE="1"は前回の送信データが完了していないことを意味しますので, TE="0"になるまで待たなければなりません. TE="0"になったら, 送信したいデータをCSI/Oトランスミットレシーブデータレジスタへライトします. この後でTE="1"にセットします. TE="1"になると送信データがTXS端子からCKSクロックの立下りエッジに同期して出力されます. すなわち, 内部クロックでボーレートを作成している場合

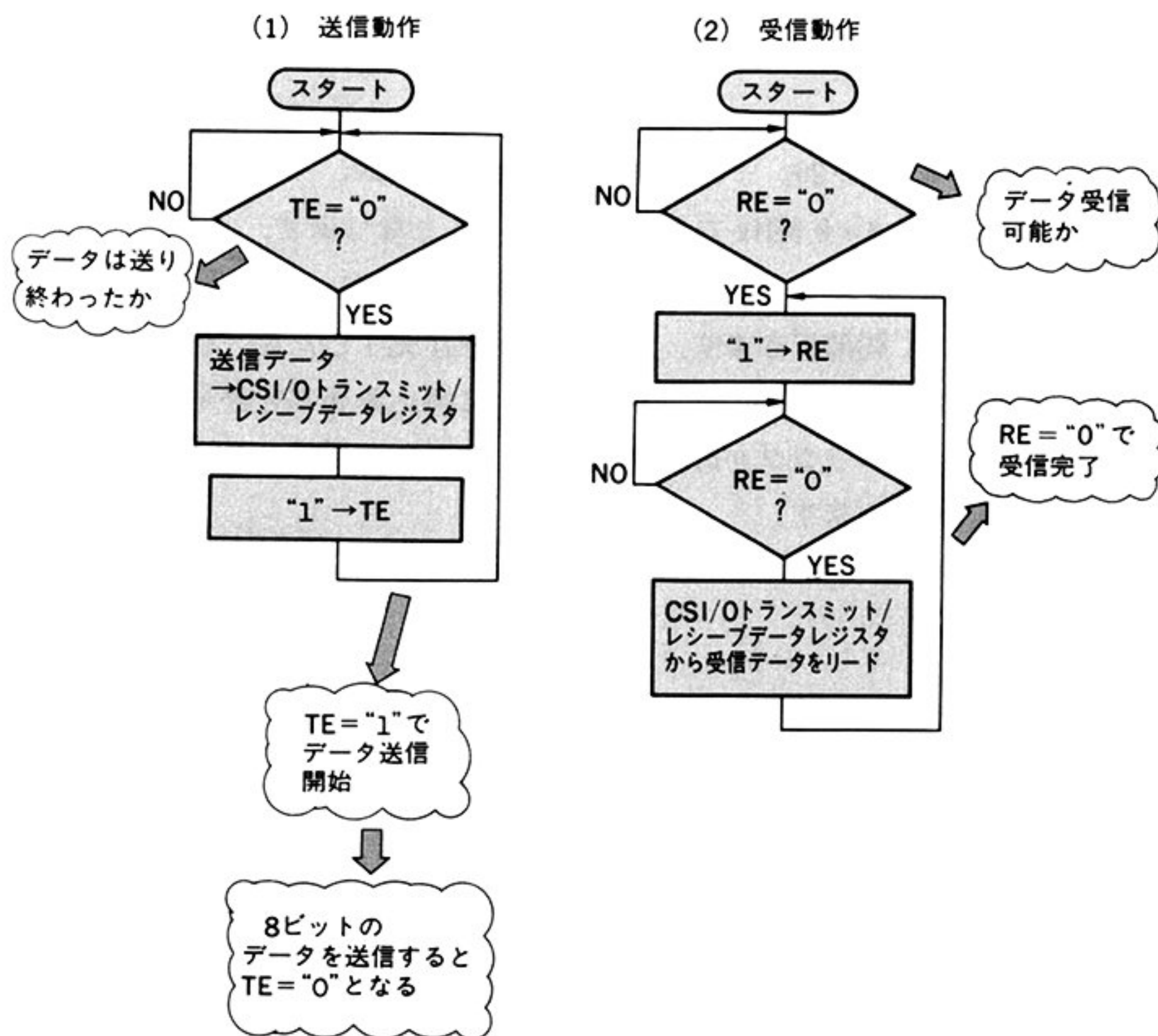



図7・2 送/受信フローチャート

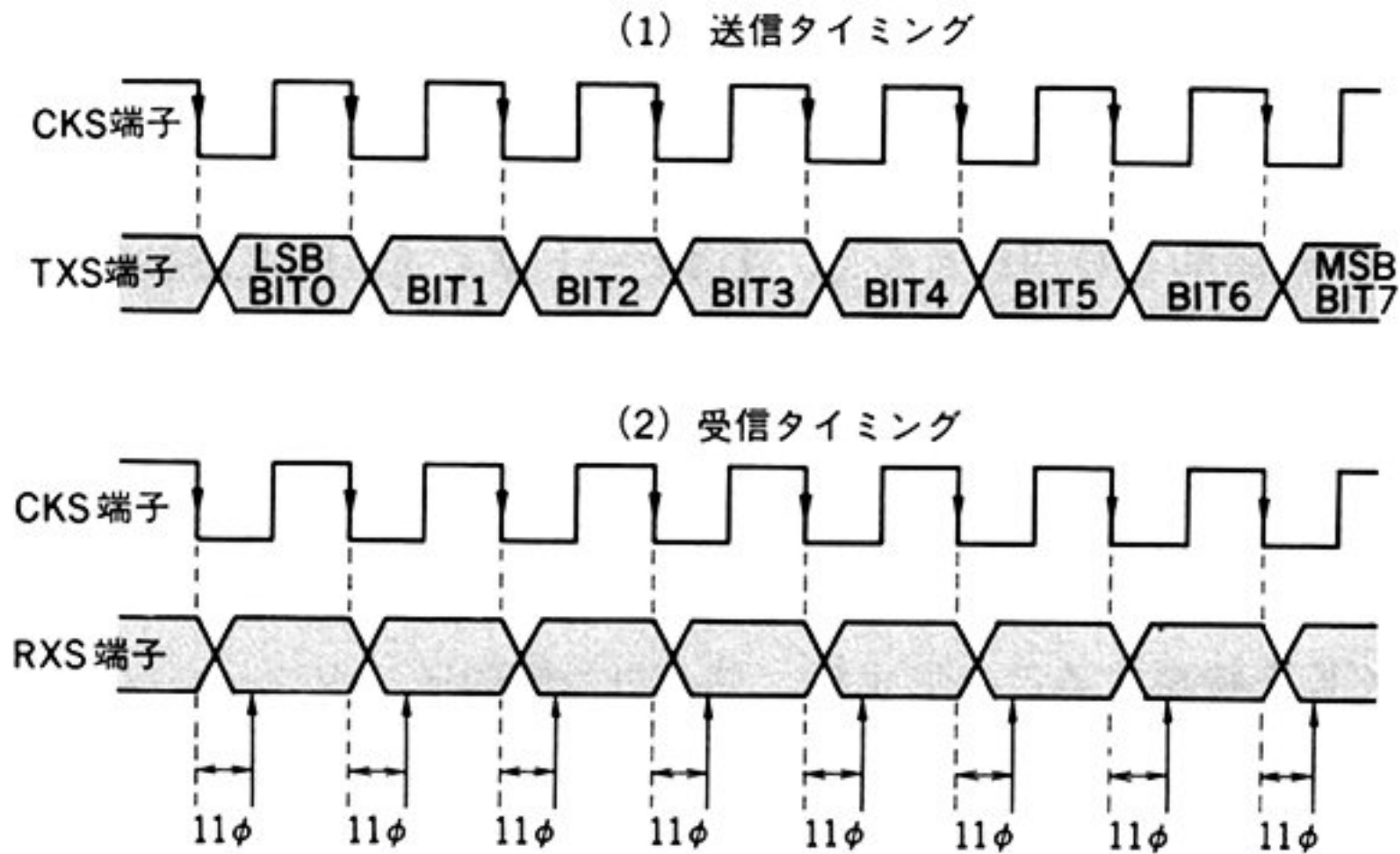



図 7・3 送/受信タイミング

は，CKS 端子からクロックが8周期分出力され，このクロックに同期して送信データがTXS 端子より出力されます．一方，外部クロックでボーレートを作成している場合は，CKS 端子より外部クロックが入力されると，それに同期してデータ送信されます．8ビットのデータ送信終了後はTE＝“0”にクリアされますので，次のデータを送ることができます．これらの処理の繰返しで送信を続けることができます．

〔2〕 **受 信 動 作**　受信動作を始めるときは，RE（レシーブイネーブル）ビットが“0”であるか確認します．RE＝“1”は前回のデータの受信が完了していないことを意味しますので，RE＝“0”になるまで待ちます．RE＝“0”の後，RE＝“1”にすると受信を開始します．つまり，送信動作同様CKS 端子のクロックに同期してRXS 端子の状態がCSI/Oデータレジスタに取り込まれ，8ビットの受信が終了するとRE＝“0”にクリアされます．RE＝“0”を確認したらCSI/Oトランスミット/レシーブデータレジスタから受信データをリードします．次のデータを受信するときは，RE＝“1”にすることにより繰返し受信処理を続けることができます．

7・3 CSI/Oの使用例

CSI/Oの具体的な使用例として，①4ビットマイコンHMCS400シリーズとのインタフェース，②LCDドライバHD61100とのインタフェースを示します．

〔1〕 **4ビットマイコンHMCS400とのインタフェース**　4ビットマイコンHMCS400シリーズは，64180と同等機能のクロック同期方式シリアルI/Oを内蔵していますので，**図7・4**に示すようにTXS(64180)→SI(HMCS400)，RXS←SO，CKS⇆$\overline{\text{SCK}}$を接続することにより，マイコン間のインタフェースが可能となります．HMCS400シリーズはI/Oポート，LCDドライバ，蛍光表示管ドライバなど豊富な機能を内蔵していますので，64180に内蔵していない機能を拡張したい場合に有効です．

〔2〕 **LCDドライバHD61100とのインタフェース**　マイコン周辺LSIでもクロック同期方式シリアルI/Oを内蔵しているデバイスがたくさんあります．たとえば，LCDドライバHD61100と64180を接続する場合，**図7・5**に示すように

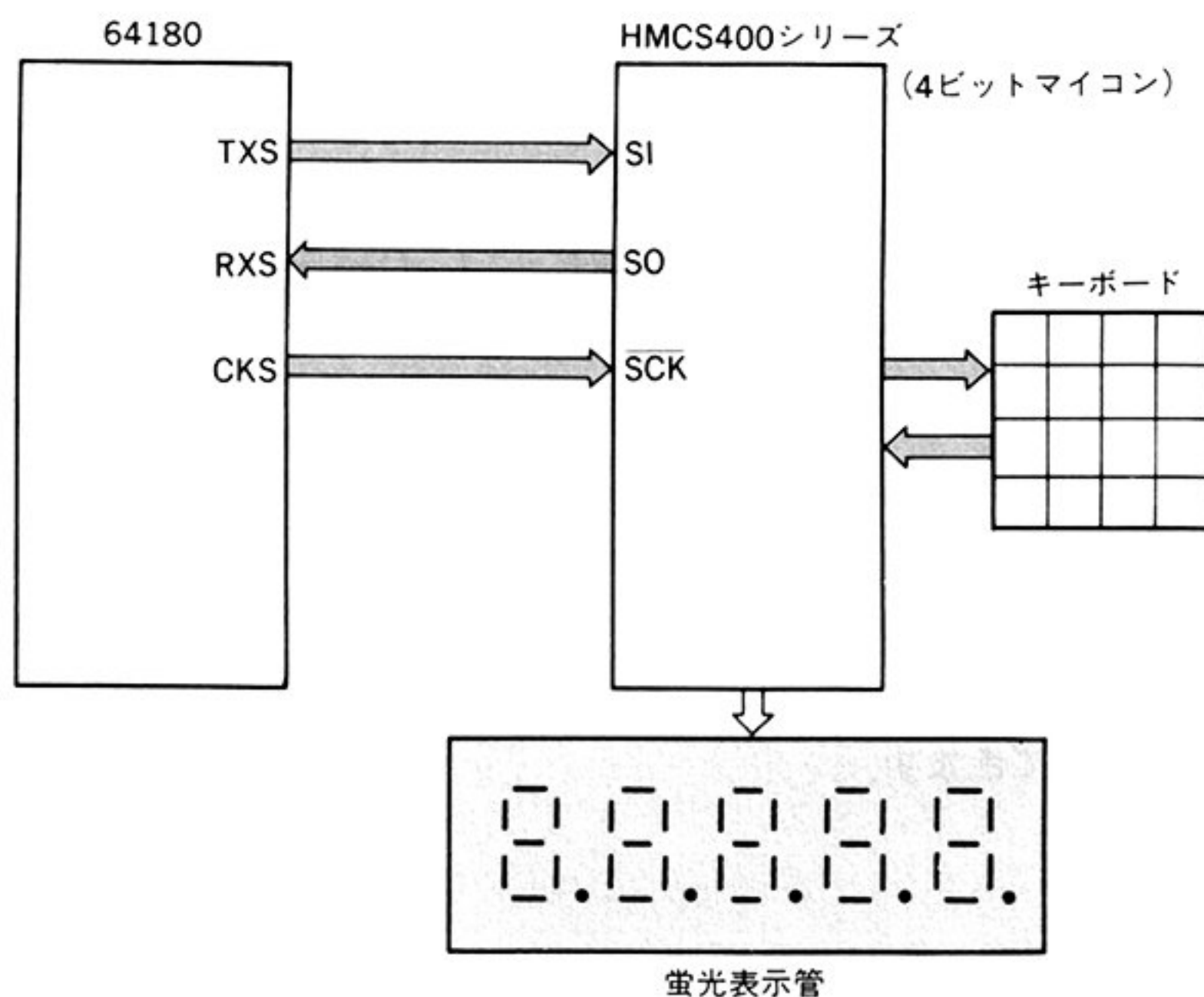



図7・4　64180と4ビットマイコンとのインタフェース

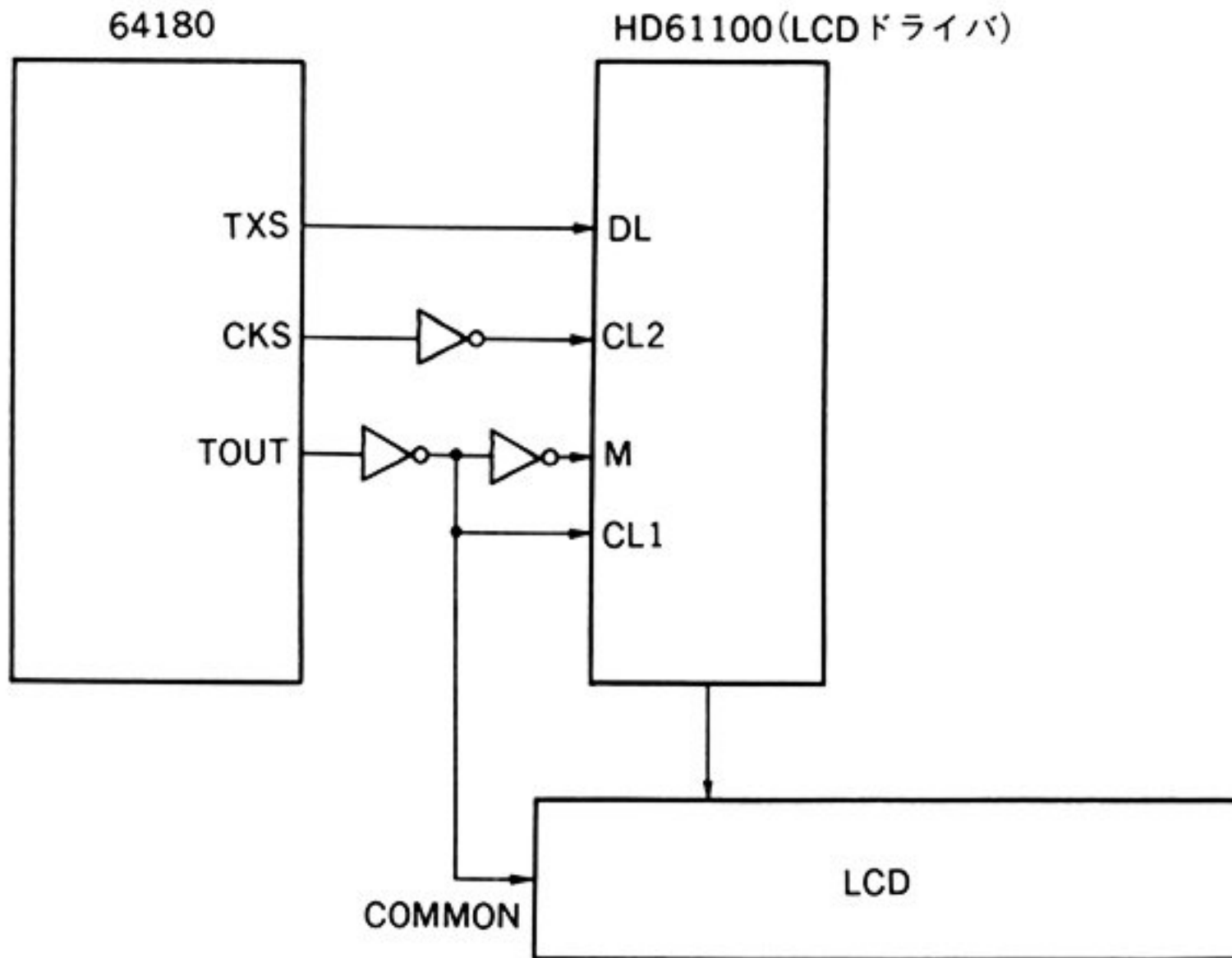



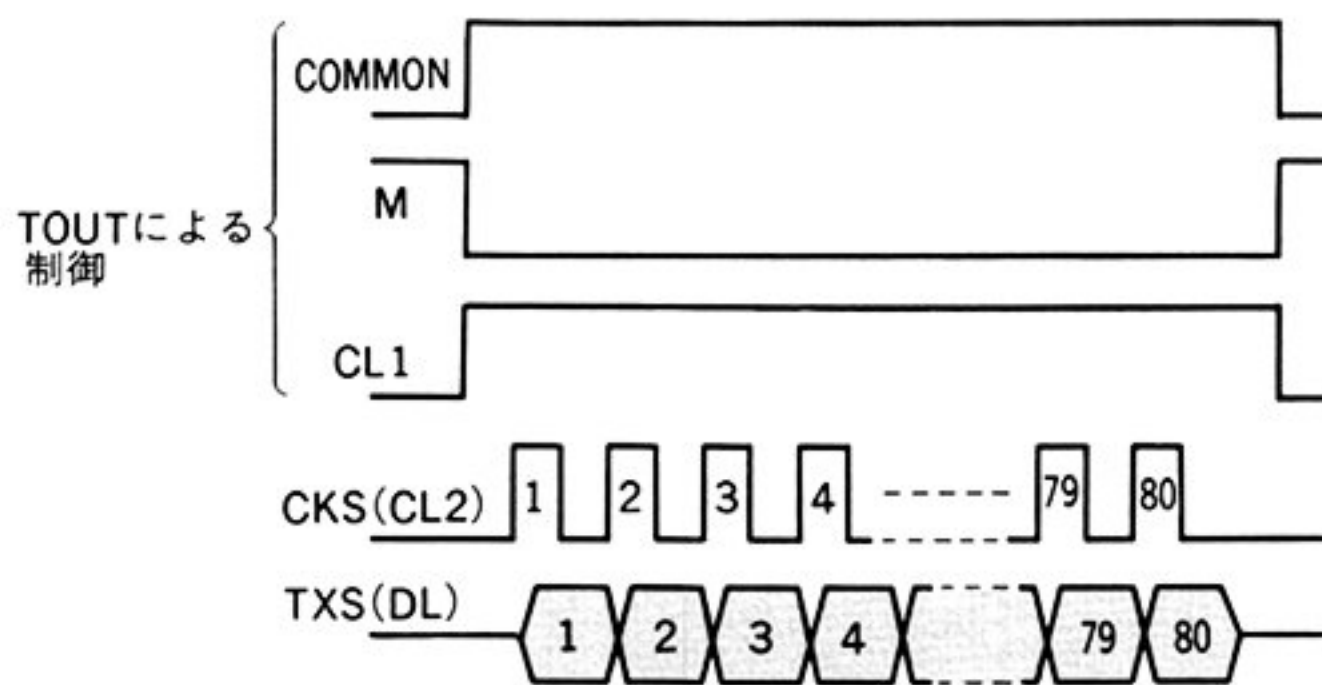
図 7・5 64180とLCDドライバとのインタフェース

```
LCDDP : LD     A,(HL)      送信したいセグメントデータのロード
        INO    C,(CNTR)    CNTRのリード
        BIT    TE,C        TE="0"のチェック
        JR     NZ,LCDDP    TE="1"だとLCDDPへ
        OUTO   (TRDR),A    TRDRへデータをライト
        LD     A,10H     } TE="1"をセットし送信開始
        OUTO   (CNTR),A  }
```

図 7・6 64180とHD61100のインタフェースタイミングおよびソフトウェア

7. クロック同期方式シリアルI/O

TXS(64180)→DL(HD61100), CKS→CL2を接続するとクロック同期方式のシリアル通信ができます．また，HD61100はLCDを表示するための交流信号MとラッチクロックCL1が必要ですので，これには64180のタイマ出力TOUTを用います．

64180から出力する信号のタイミングは，**図7•6**に示すようにTOUT，CKS，TXSで決まります．このうち，CSI/Oで制御するCKS端子からのクロック送出およびTXS端子からのデータ送信の手順を，ソフトウェア例をみながら説明します．

手順は，① 送信したいセグメントデータのロード，② TEビットのチェック，③ TRDR（トランスミット・レシーブ・データ・レジスタ）へ送信データのライト，④ TE＝“1”にセットとなります．このように7ステップという短いプログラムで8ビットのデータ送信が可能となります．80セグメントのLCDを表示する場合は，この基本ルーチンを10回繰り返しCL1を立ち下げることにより表示データが変更可能となります．

8. WAIT リフレッシュ コントローラ

64180のCPUは低速のメモリやI/O機器をアクセスする場合に，バスサイクルを引き延ばしています．また，DRAMとインタフェースするためには，リフレッシュが必要です．従来，これらの機能は外付回路にて実現していましたが，64180にはこれらのいわゆる雑ロジックを内蔵しているため，メモリやI/Oとのインタフェース回路が簡単になります．

8・1 ウェイトの動作原理と使用法

〔1〕**概　　要**　64180には低速のメモリやI/Oのアクセスのために，リードまたはライトサイクルを引き延ばすためのウェイトステート挿入機能があります．ウェイトは1クロック単位で与え，T_2ステートとT_3ステートとの間にダミーのステートとしてウェイトステートT_Wが挿入されて，そのサイクルを引き延ばしています．ウェイトステートの挿入は，ハードウェアによる方法（$\overline{\mathrm{WAIT}}$端子への0レベル入力）とソフトウェアによる方法（ウェイトコントロールレジスタのビット設定）とがあります．

〔2〕**ハードウェアによる方法**　この方法は，**図8・1**に示すようにT_2ステートのϕクロックの立下りのタイミングで，$\overline{\mathrm{WAIT}}$端子の状態をサンプリングし，次のように動作します．

$\overline{\mathrm{WAIT}}$端子が“L”のとき……T_Wステートが挿入される
$\overline{\mathrm{WAIT}}$端子が“H”のとき……T_Wステートは挿入されず
　　　　　　　　　　　　　　　T_3ステートへ移る

引き続いてT_Wステートを2個以上挿入する場合は，$\overline{\mathrm{WAIT}}$端子を必要な数だけ“L”にします．

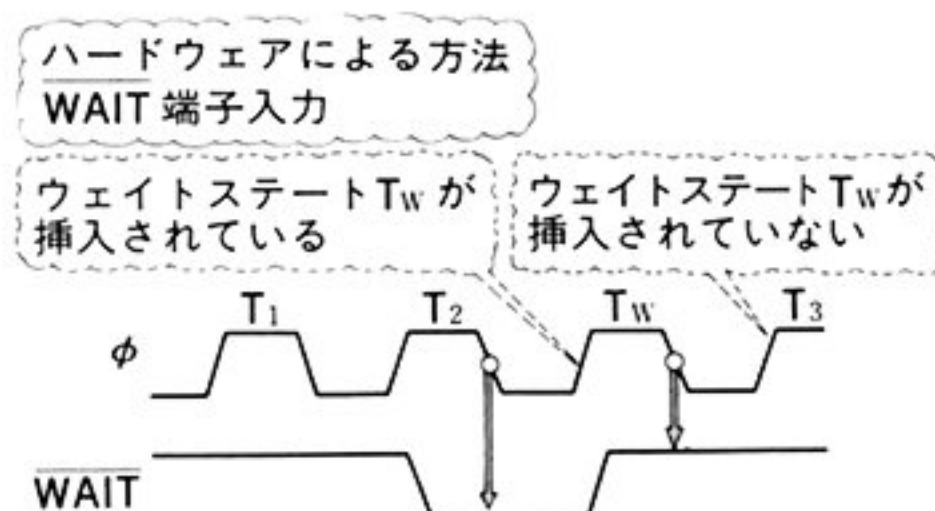



図8・1　$\overline{\mathrm{WAIT}}$端子によるウェイトステートの挿入

〔3〕**ソフトウェアによる方法**　ウェイトステートコントロールレジスタのビットを設定することにより，規定数のT_Wステートを挿入する方法です（**図8・2**参照）．図のように，メモリ空間とI/O空間とは独立してT_Wステート数を設定できるので，それぞれのアクセスタイムに最適なT_Wステート数が選べます．メモリ空間では0～3，I/O空間では1～4です．T_Wステート数は，それぞれの空間の

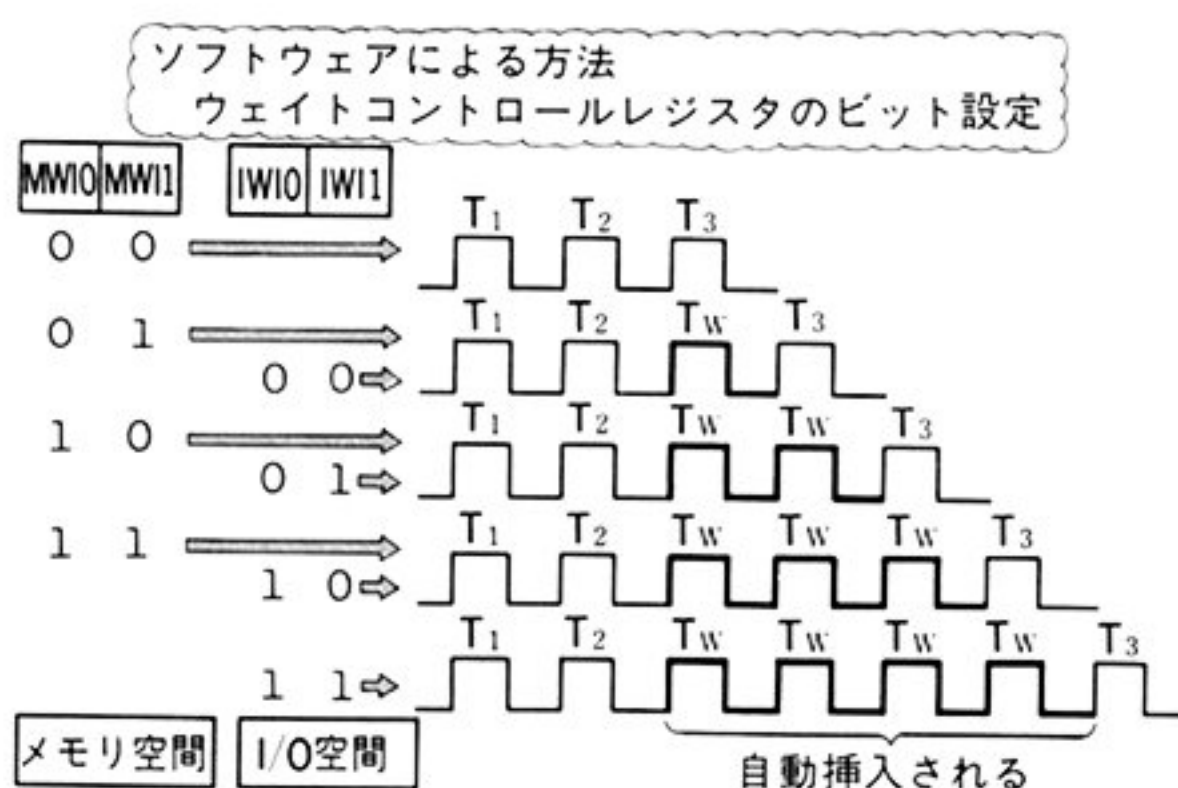



図 8・2 ウェイトコントロールレジスタによるウェイトステートの挿入

中では一定です．したがって，たとえばアクセスタイムの遅い EPROM と速い SRAM が混在する場合，EPROM に合わせて T_W ステート数を設定することになります．ウェイトステートコントロールレジスタは，リセット後には最大 T_W ステート数となるように初期化されます．また，そのレジスタは DMAC チャネル 1 のコントロールレジスタと I/O アドレスが同じです．

〔4〕 **ソフトウェアとハードウェアの同時要求**　T_Wステート挿入要求がソフトウェアとハードウェアともに同時の場合には，長いほうに合わせて挿入されます．

8・2 リフレッシュコントローラの動作原理

〔1〕 **概　　要**　64180では，DRAMリフレッシュはリフレッシュサイクルという特別なマシンサイクルにて行われます．Z80ではオペコードフェッチサイクルに付随していますが，64180ではオペコードフェッチサイクルとは独立しています．また，64180ではリフレッシュサイクルを挿入するかしないか，リフレッシュサイクルの挿入の間隔，リフレッシュサイクル長を2ステートか3ステートかを選択できます．

〔2〕 **リフレッシュサイクル**　マシンサイクルの切れ目で挿入されます．図8・3の例では命令実行中に挿入されていますが，DMA実行中でもマシンサイクルの切れ目に挿入されます．図8・3のリフレッシュ要求の間隔は10ステートを選択しています．しかしながら，図8・3では「OUTO (3F)，A」命令のI/Oライトサイクル後に挿入されるため，10ステートより長くなったり，あるいは短くなったりします．一方，リフレッシュサイクルが挿入されないのは次の場合です．リ

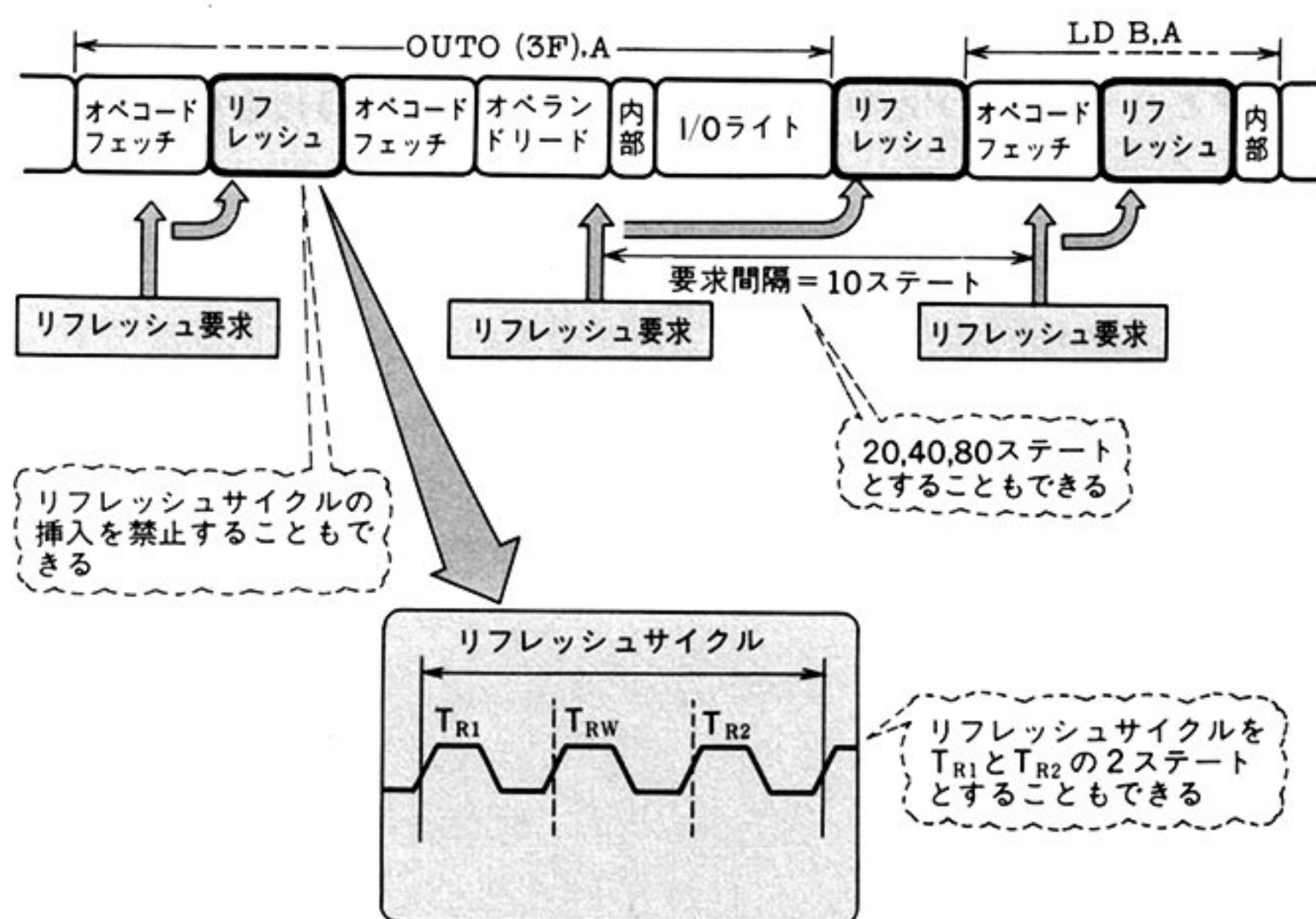



図 8・3　リフレッシュサイクルの挿入動作

セット，バスリリースモード，スリープおよびシステムストップモード，ウェイトステートが長く挿入されている場合です．このようなモードのときにDRAMのデータが消失しては困る場合には，別の手段でリフレッシュされるような工夫をします．

〔3〕 **リフレッシュアドレス**　64180では8ビットです．256K DRAMのような256リフレッシュサイクルのDRAMには，外部回路を付加せずに使用できます．なお，CPUの専用レジスタのRカウンタはZ80ではリフレッシュアドレスですが，64180ではリフレッシュアドレスとは一致していません．オペコードフェッチごとにカウントアップするカウンタです．

〔4〕 **リセット後**　リフレッシュアドレスは00Hからスタートして，自動的にリフレッシュサイクルが挿入されます．挿入間隔は最短の10ステートで，リフレッシュサイクルは3ステートです．その後，動作周波数とメモリリフレッシュ間隔により最適値に設定することになります．また，DRAMを使っていなければリフレッシュサイクルの挿入を禁止します．

8・3 リフレッシュタイミングとDRAMとのインタフェース

〔1〕 **タイミング**　リフレッシュサイクルは，T_{R1}，(T_{RW})，T_{R2}の2または3ステートで構成されています．T_{RW}はリフレッシュウェイトステートで，レジスタ設定によりソフトウェアで挿入することができます（**図8•4**）．

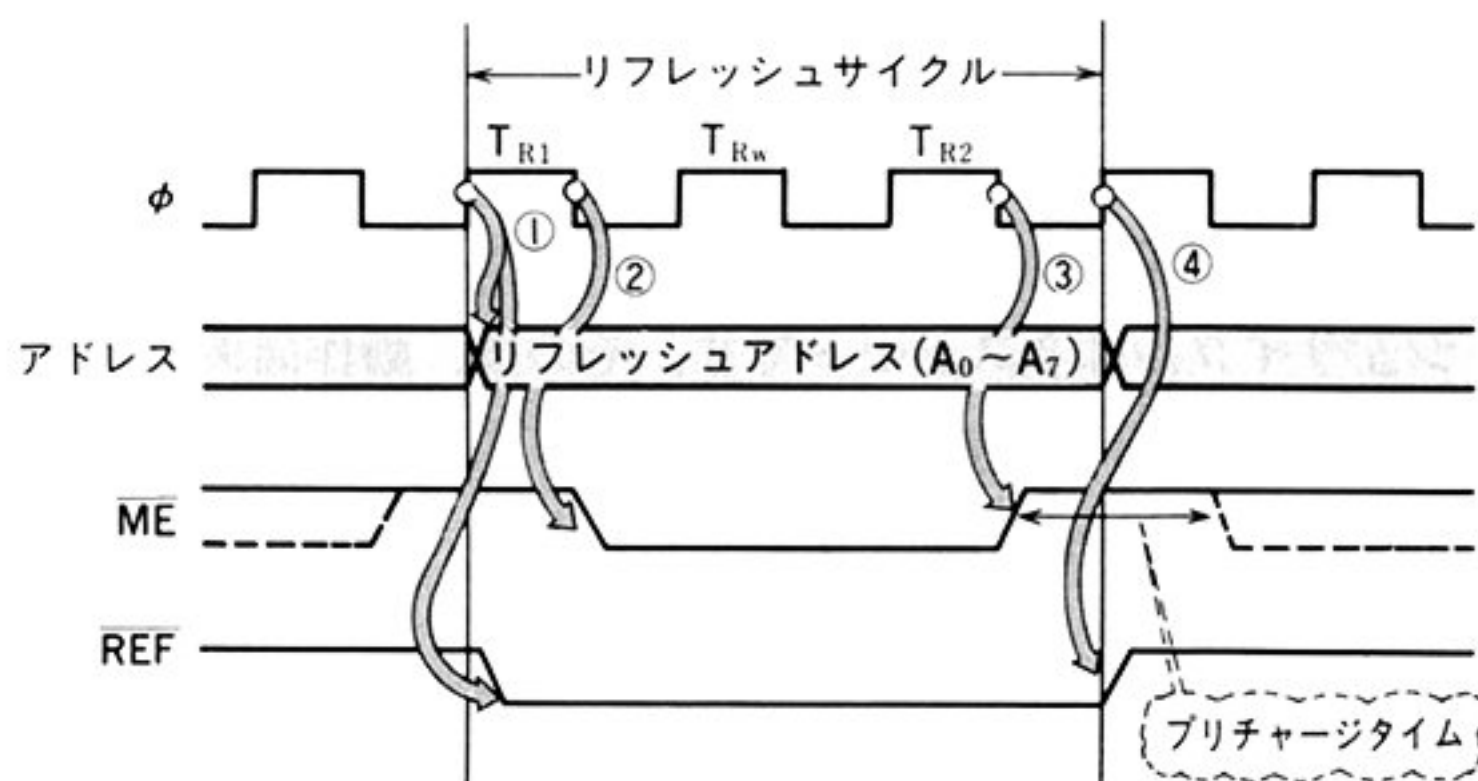



図 8・4　リフレッシュサイクル（リフレッシュウェイトを挿入している場合）

このサイクルのシーケンスは次のとおりです．①リフレッシュT_1ステート(T_{R1})の前半で，リフレッシュアドレスが出力されます．同時に，$\overline{\text{REF}}$信号は“L”となります，②T_{R1}ステートの後半で，$\overline{\text{ME}}$はアクティブとなりリフレッシュの起動がかかります，③リフレッシュT_2ステート(T_{R2})の後半で，$\overline{\text{ME}}$はインアクティブとなりますが，リフレッシュアドレスは次のバスサイクルまで保持されています，④$\overline{\text{REF}}$信号は次のサイクルの先頭でインアクティブとなります．

〔2〕 **DRAMとのインタフェース**　64180はリフレッシュコントローラを内蔵しているため，**図8•5**のようにDRAMのデータ，$\overline{\text{RAS}}$，$\overline{\text{WE}}$端子には直結できます．一方，メモリアドレスと$\overline{\text{CAS}}$信号はそれぞれ外部回路で生成します．具体的なインタフェース例については，第III編を参照してください．

図 8・5　DRAMとの接続例

9. 割　込　み

64180は強力な割込み機能をもっています．割込み要因としては，外部4本，内部8本の計12本の要因があります．このうち，外部割込みはZ80がもっていた$\overline{\text{NMI}}$と$\overline{\text{INT0}}$で，これに加え64180で追加された$\overline{\text{INT1}}$，$\overline{\text{INT2}}$の2本の割込みがあります．また，内部割込みはZ80にはなかった割込みで内蔵の周辺（DMAC，タイマ，CSI/O，ASCI）の割込みおよびTRAP割込みです．

9・1　割込みとは

〔1〕 **割込みの概念**　割込みとは，通常のシーケンシャルに実行するプログラムより優先度が高く，かつ緊急に処理する必要のあるプログラムを優先して実行するための手法です．CPU は，割込みの要求があるとそこまで実行していたプログラムを中断し，割込みの要求に応じたプログラムを実行します（**図 9・1**）．

また，割込みには，ソフトウェアでマスクできないノンマスカブル割込み（要求が入ると必ず受け付けられる割込み）と，割込みを受け付けるかどうかをソフトウェアで制御可能なマスカブル割込みがあります．

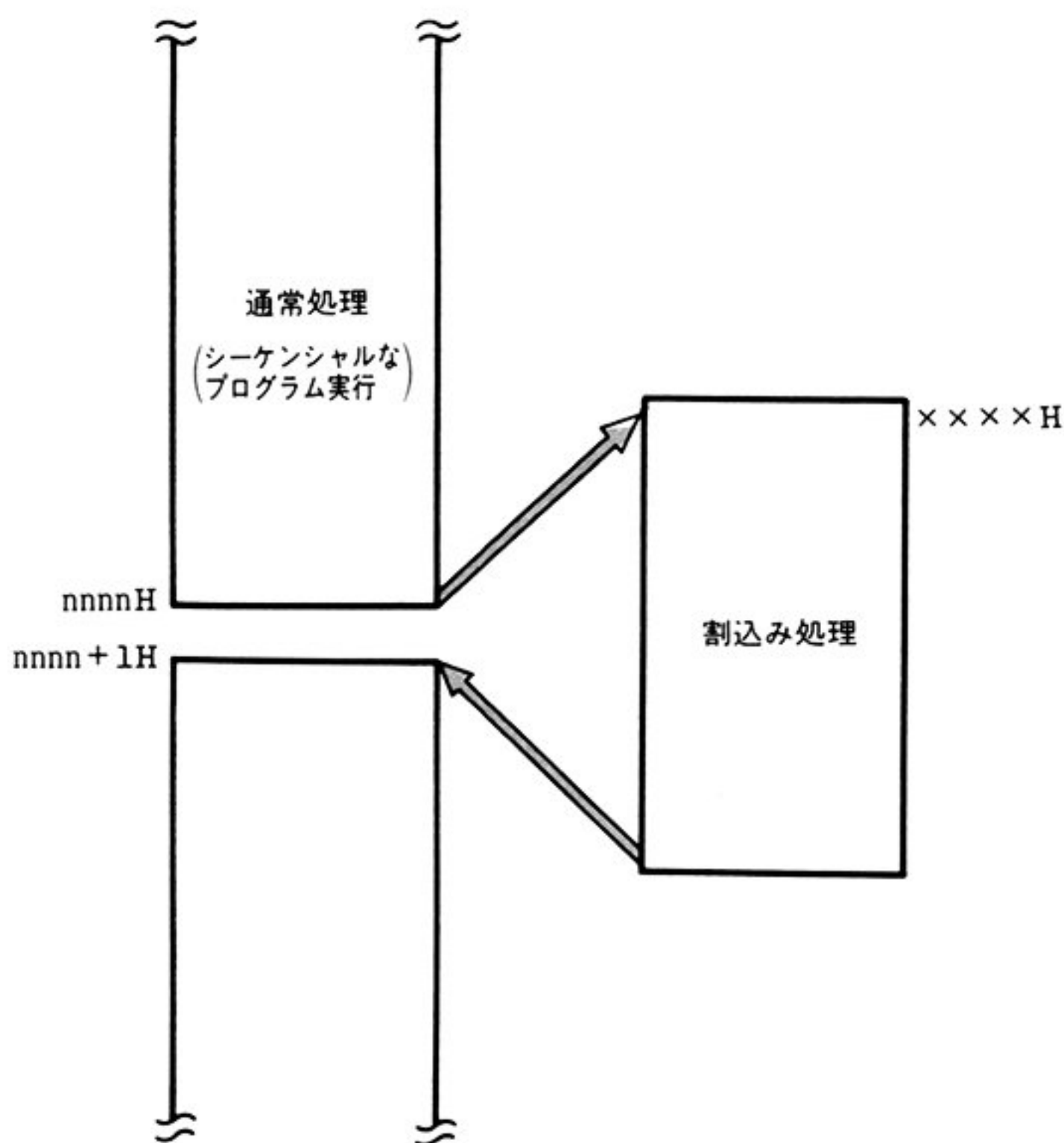



図 9・1　割込み処理

〔2〕 **64180 の割込み要因**　64180 の割込み要因としては，外部（割込みの要因が 64180 の外部に入り割込み端子から要求が入る）4 本と，内部（割込みの要因が 64180 の内部にある）8 本の計 12 本の要因があります．

外部割込みの要因としては，ノンマスカブルな $\overline{\text{NMI}}$ があります．$\overline{\text{NMI}}$ が発生すると 66 番地へジャンプします．これ以外に，マスカブルな $\overline{\text{INT}}$ 3 本（$\overline{\text{INT0}}$，$\overline{\text{INT1}}$，$\overline{\text{INT2}}$）があります．このうち $\overline{\text{INT0}}$ は，Z80 がもっていたモード 0 ～ 3 の 3 つの割込みモードをもっています．$\overline{\text{INT1}}$ および $\overline{\text{INT2}}$ は 64180 で追加された外部割込み要因で，ベクタ方式の割込みです．

次に内部割込み要因としては，TRAP，タイマ 2 本，DMAC 2 本，CSI/O 1 本，ASCI 2 本の計 8 本があります．このうち TRAP 割込みは，未定義のオペコードをフェッチすると割込みが発生するシステムのフェールセーフのための機能です．TRAP が発生すると 0 番地へのジャンプが発生します．なお TRAP はソフトウェアでマスクできません．

また，タイマ（2），DMAC(2)，CSI/O，ASCI(2) の計 7 つの要因は，$\overline{\text{INT1}}$，$\overline{\text{INT2}}$ と同様なベクタ方式の割込みです．

なお，これらの割込みには優先順位があり，**表 9・1** に示すとおり，TRAP，$\overline{\text{NMI}}$，$\overline{\text{INT0}}$，$\overline{\text{INT1}}$，$\overline{\text{INT2}}$，タイマ 0，タイマ 1，DMAC0，DMAC1，CSI/O，ASCI0，ASCI1 の順番の優先度となっています．

表 9・1 割込みの種類と優先順位

優先順位	割込み要因	内部/外部	備 考
1	TRAP	内 部	ベクタ（0000H 固定）
2	$\overline{\text{NMI}}$	外 部	ベクタ（0066H 固定）
3	$\overline{\text{INT0}}$		モード 0～2
4	$\overline{\text{INT1}}$		ベクタ方式（プログラマブル）
5	$\overline{\text{INT2}}$		
6	タイマ 0	内 部	
7	タイマ 1		
8	DMA チャネル 0		
9	DMA チャネル 1		
10	CSI/O		
11	ASCI チャネル 0		
12	ASCI チャネル 1		

9・2 ベクタ方式

〔1〕 **ベクタ方式**　ベクタ方式とは，割込みの処理開始番地を決定するための手法です．ベクタ方式の割込みの場合，まず割込み要因ごとに割り当てられた割込みベクタアドレスのメモリからリードを行います．この割込みベクタアドレスには，割込み処理ルーチンの開始アドレスが2バイトにわたって格納されており，CPUはこの開始アドレスから割込み処理ルーチンを実行します．したがってベクタアドレスに配置されているこの割込み処理ルーチンの開始アドレスを設定することにより，メモリ空間のすべてのエリアに割込み処理ルーチンをおくことが可能となります（**図9・2**）．

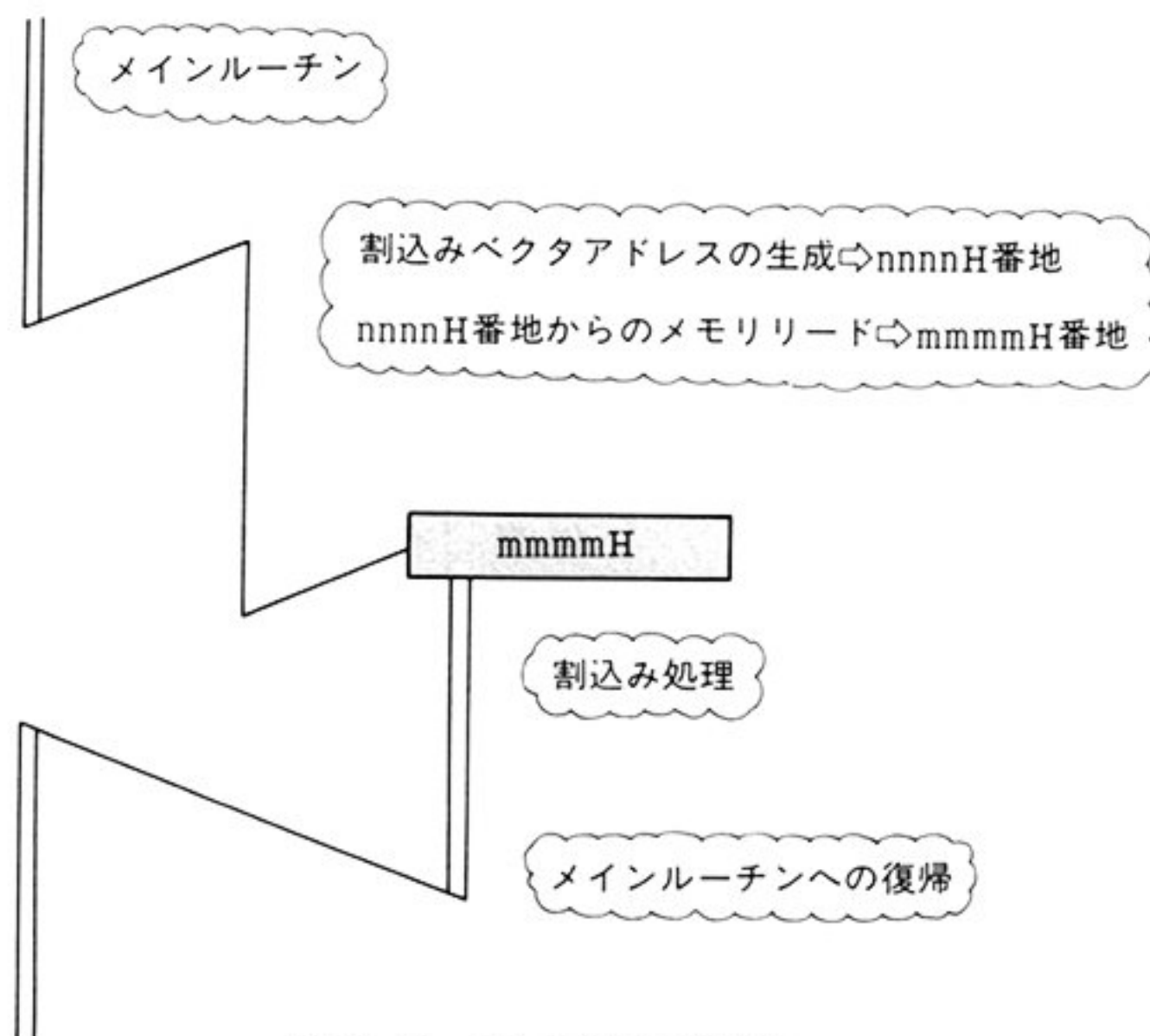



図 9・2　ベクタ方式の割込み

〔2〕 **64180のベクタ割込み**　64180の割込みのうち$\overline{\mathrm{INT0}}$（モード2），$\overline{\mathrm{INT1}}$，$\overline{\mathrm{INT2}}$，TRAPを除く内部割込み（タイマ，DMAC，CSI/O，ASCI）は，ベクタ方式の割込みを行います．

$\overline{\mathrm{INT0}}$モード2のベクタは，上位8ビットにIレジスタを使用し，下位8ビットに割込みアクノレッジサイクル中に，データバスから読み込んだ8ビットのデータを使用します．

次に，$\overline{\text{INT1}}$，$\overline{\text{INT2}}$および内部割込み（TRAPを除く）の割込みベクタアドレスは次のように生成されます．上位8ビットはIレジスタにより与えられます．下位8ビットのうち，その上位3ビットはILレジスタ（I/O空間の33H番地に配置）により与えられ，下位5ビットは要因ごとに固定コードとして与えられます．ベクタアドレスの生成の様子と固定コードを**図9・3**に示します．

こうして生成された16ビットのベクタアドレスから割込みルーチンの開始番地のリードを行い，おのおのの割込み処理ルーチンを実行します．

	固定コード	備考
$\overline{\text{INT1}}$	00000	外部割込み
$\overline{\text{INT2}}$	00010	〃
タイマ0	00100	内部割込み
タイマ1	00110	〃
DMAC0	01000	〃
DMAC1	01010	〃
CSI/O	01100	〃
ASCI0	01110	〃
ASCI1	10000	〃

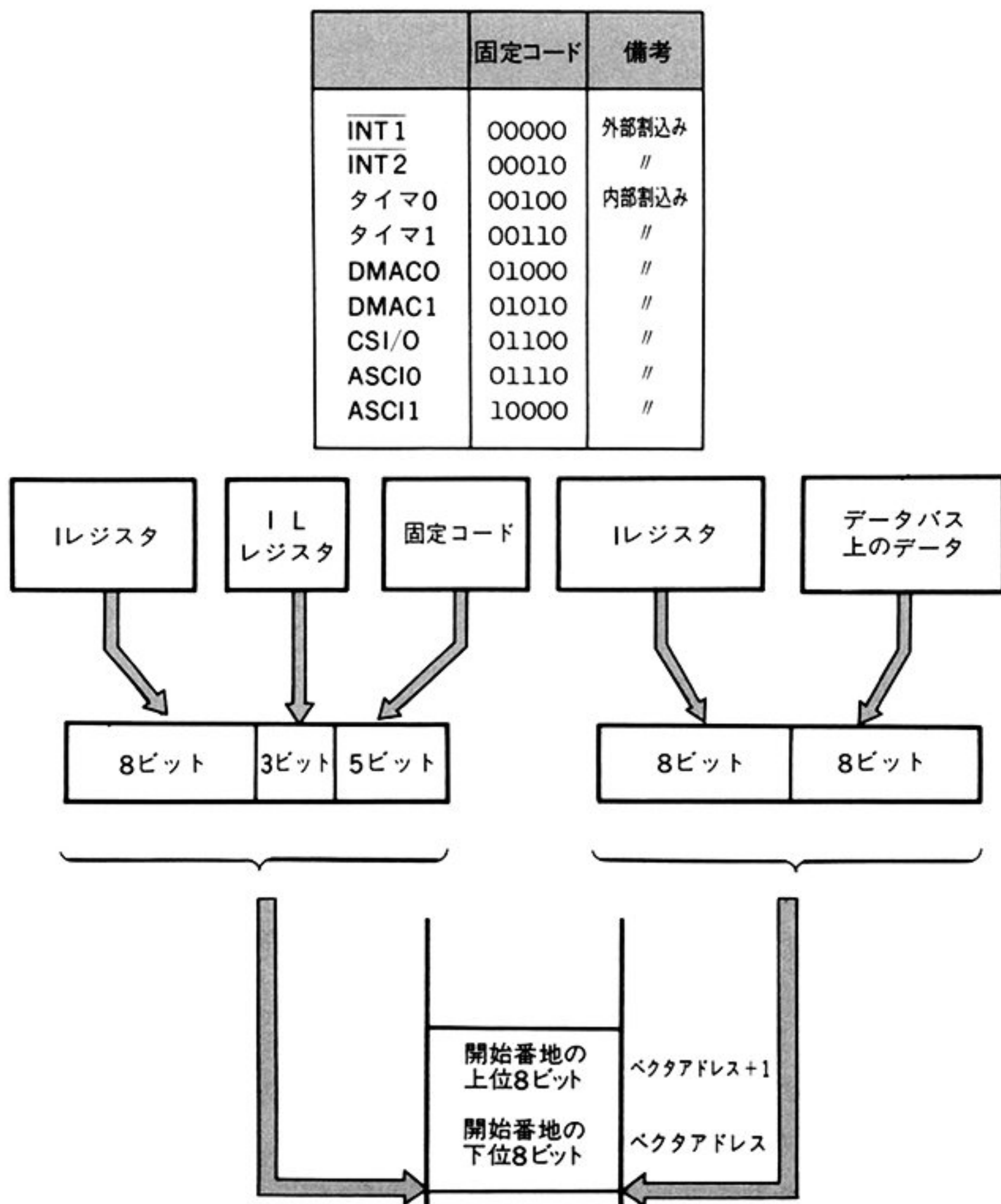



図9・3 64180のベクタ方式の割込み

9. 割　込　み

$\overline{INT0}$ モード2の場合，ベクタを生成するため外部にハードウェアが必要となります．Z80の周辺LSIを使用すればこの機能はサポートされていますが，68系の周辺LSIやインテルの80系周辺LSIを使用する場合は，外付回路で対応することになります．これに対し，$\overline{INT1}$，$\overline{INT2}$などのベクタアドレスは自動的に生成します．したがって，68系やインテルの80系の周辺LSIを使用する場合は，$\overline{INT1}$，$\overline{INT2}$を使用するのが便利です．

IEFフラグ

IEFフラグは，直接CPUからリードすることが可能です．LD　A，Iあるいは LD　A，R命令を実行することにより，IEF_2の内容がP/Vフラグに転送され，IEFフラグの内容を知ることができます．ここで注意すべき点は，Z80ではLD　A，IあるいはLD　A，R命令の実行中に割込みが発生した場合，割込み発生後のIEFフラグがP/Vフラグに反映され，正しい値が読み出せなくなります．そこで，64180(R1/Z)ではLD　A，IとLD　A，R命令の最後では，割込みをサンプリングしません．したがって，これらの命令を実行すると，IEFフラグの正しい値がP/Vフラグに反映されます．

9・3 マスカブルな割込み

〔1〕 **IEF フラグ** 64180はソフトウェアで割込み禁止可能なマスカブル割込みとして，内部7本，外部3本の要因があります．これらのマスカブルな割込みは，**IEF** (Interrupt Enable Flag) **フラグ**により制御されています．IEFフラグはリセット直後‘0’に初期化されており，この状態ではマスカブルな割込みを受け付けることはできません．IEFフラグを‘1’にセットし，マスカブルな割込みを受付け可能とするにはEI命令を実行する必要があります．また，IEFフラグはDI命令により‘0’にリセットすることも可能です．また，マスカブルな割込みが受け付けられるとIEFフラグが‘0’にリセットされ，他の割込みが受付け不可能となります．再度割込み受付け可能とするためには，EI命令を実行します．一般に，EI命令は割込み処理ルーチンの終了するRETI命令の直前に実行します．

〔2〕 **$\overline{\text{INT}}$** 64180は，外部入力の3本の割込み要求端子をもっています．この3本の$\overline{\text{INT}}$端子（$\overline{\text{INT0}}$，$\overline{\text{INT1}}$，$\overline{\text{INT2}}$）は，IEF以外に個別に割込み許可フラグをもっています．このフラグは，I/O空間上に配置されたINT/TRAPコントロールレジスタに配置されています．このフラグはそれぞれITE0，ITE1，ITE2と呼ばれ，リセット後‘100’に初期化されます．したがって，$\overline{\text{INT0}}$のみEI命令実行後すぐに受付け可能であり，$\overline{\text{INT1}}$，$\overline{\text{INT2}}$はマスクされた状態になっています．$\overline{\text{INT}}$端子はレベルセンスの割込みです．これらの割込み要求は，各命令の最後のマシンサイクルの後ろから2番目のϕクロックの立下りエッッ

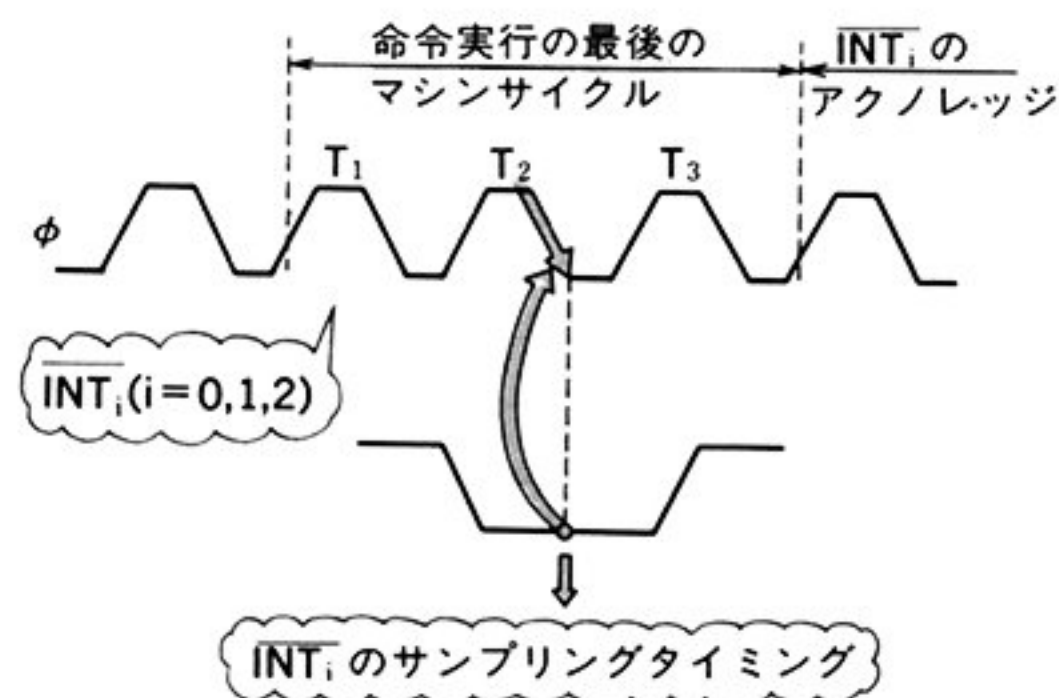



図 9・4 $\overline{\text{INT}}$のサンプリング

ジでサンプリングされています．サンプリングのタイミングを**図 9・4** に示します．

〔3〕 **内部割込み**　内蔵の周辺からの割込み要因も，IEF フラグ以外に要因別の割込み許可フラグをもっています．これらの割込み許可フラグはリセット直後‘0’に初期化されています．したがって内部割込みを発生させるには，EI 命令を実行するだけでなく，割込み要因ごとにもっている割込み許可フラグを‘1’にセットしてやる必要があります．この割込み許可フラグは，I/O 空間上の各コントロールレジスタに配置されています．これらのレジスタとフラグ名を**表 9・2** に示します．

表 9・2　内部割込みの割込み許可フラグ

周　辺	レ ジ ス タ		I/O アドレス	割込み許可フラグ	リセット直後の値
タイマ0	タイマコントロールレジスタ		10H	TIE0	0
タイマ1	タイマコントロールレジスタ		10H	TIE1	0
DMAC0	DMA ステータスレジスタ		30H	DIE0	0
DMAC1	DMA ステータスレジスタ		30H	DIEI	0
CSI/O	CSI/O コントロールレジスタ		0AH	EIE	0
ASCI0	送信	ASCI ステータスレジスタ ch0	00H	TIE(0)	0
	受信	ASCI ステータスレジスタ ch0	00H	RIE(0)	0
ASCI1	送信	ASCI ステータスレジスタ ch1	01H	TIE(0)	0
	受信	ASCI ステータスレジスタ ch1	01H	RIE(0)	0

9・4 $\overline{\text{INT0}}$モード0

$\overline{\text{INT0}}$は，Z80と同様にモード0からモード2の3つの動作モードをもっています．このうちモード0は，インテル社の8080，8085などがもっている割込みと同一の動作モードです．このモードを使用すると，たとえばインテル社の8259のような割込みコントローラを使用できます．

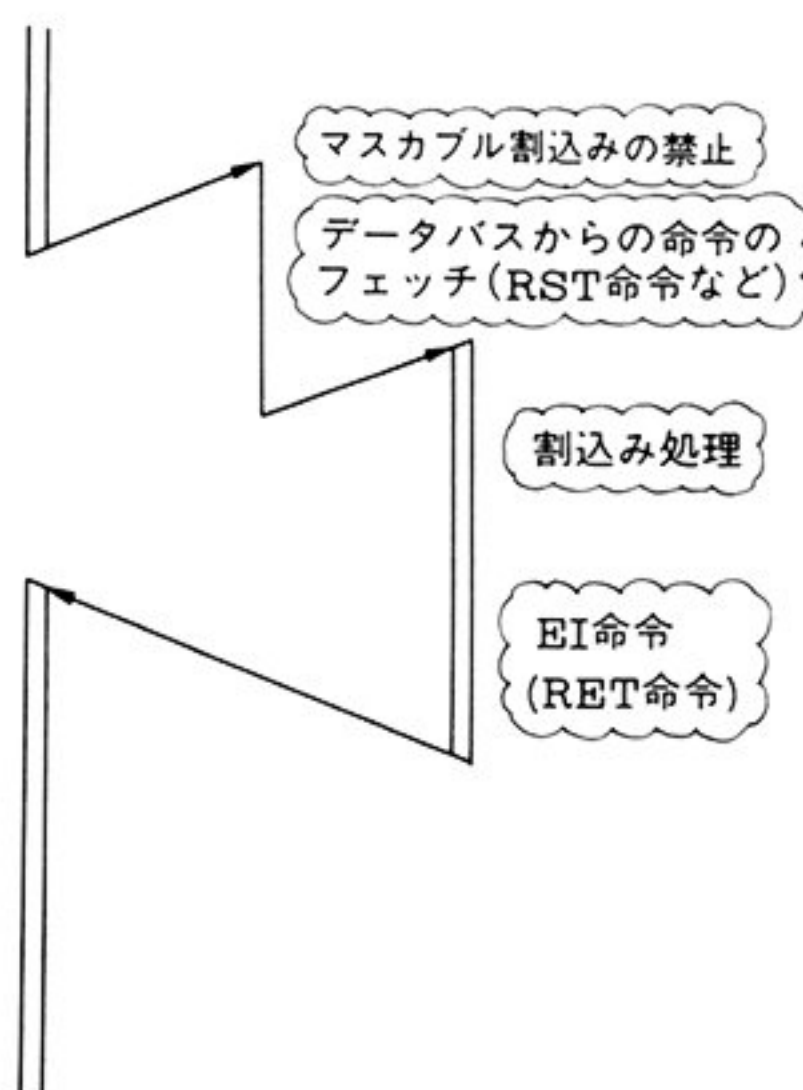



図 9・5 $\overline{\text{INT0}}$ モード0のシーケンス

このモードでは，割込みアクノレッジサイクル時にデータバス上のデータを命令としてフェッチし実行します．この命令としては，RST命令やCALL命令を使用し，そのジャンプ先に割込み処理のためのプログラムを配置します（**図9・5**）．

割込み要求（$\overline{\text{INT0}}$ 端子）がアクティブ（“Low” 入力）になると，CPUは割込みアクノレッジサイクルを実行します．このアクノレッジサイクル中には，$\overline{\text{LIR}}$信号と$\overline{\text{IOE}}$信号がアクティブになります．8259などの割込みコントローラを使用する場合，割込み要求に対してアクノレッジサイクル信号（$\overline{\text{ACK}}$信号）を返す必要があります．この$\overline{\text{ACK}}$信号を作るために，$\overline{\text{LIR}}$信号と$\overline{\text{IOE}}$信号を使用することができます．この$\overline{\text{ACK}}$信号を受けると，8259はデータバス上にCALL命令をのせてきます．64180は，割込みアクノレッジサイクルのT_3ステートの立上りで，CALL命令のオペコードを取り込み実行します．**図9・6**と**図9・7**に$\overline{\text{INT0}}$モード0

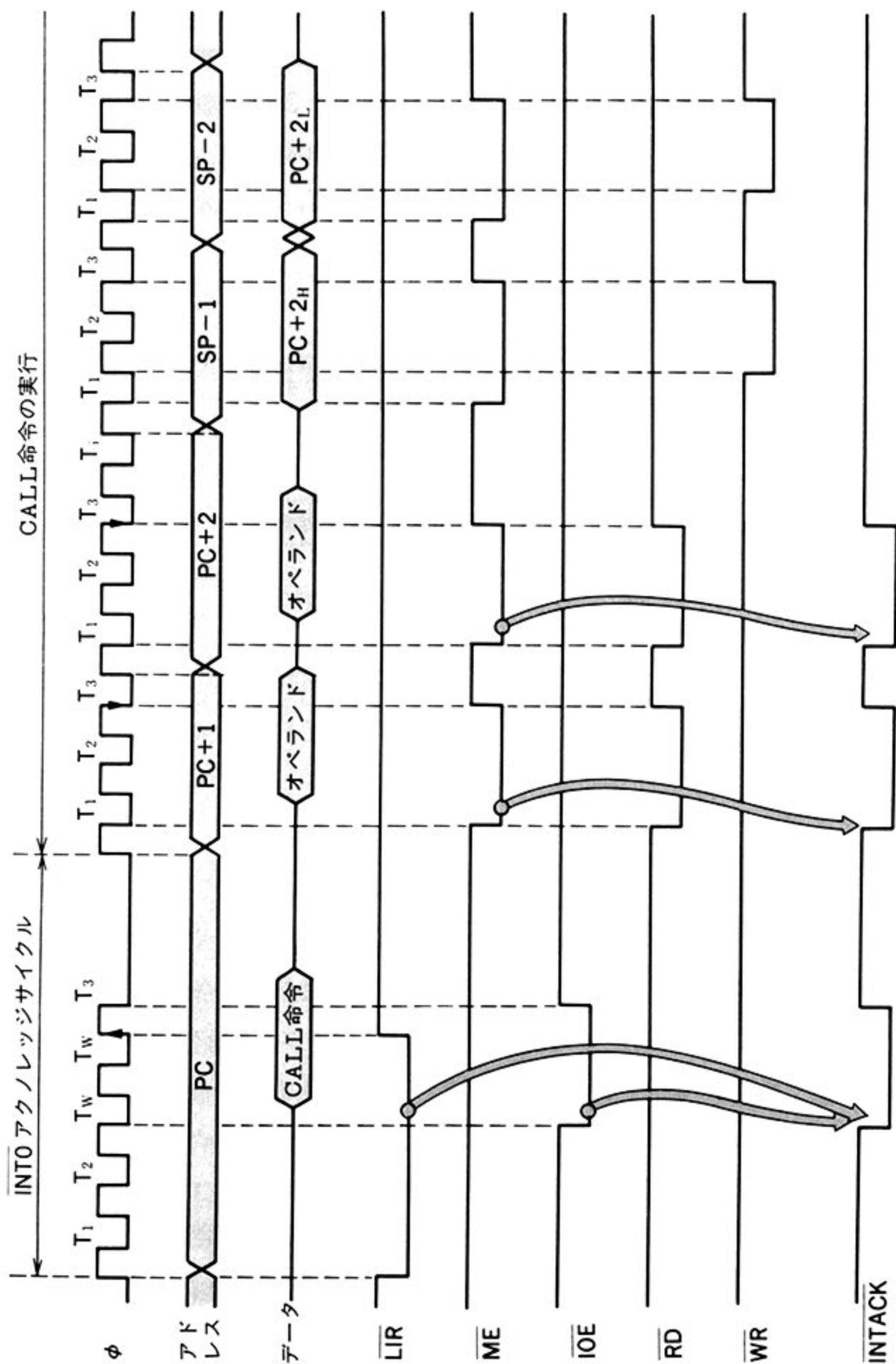



図 9・6　INT0 モード0のタイミングとINTACK信号

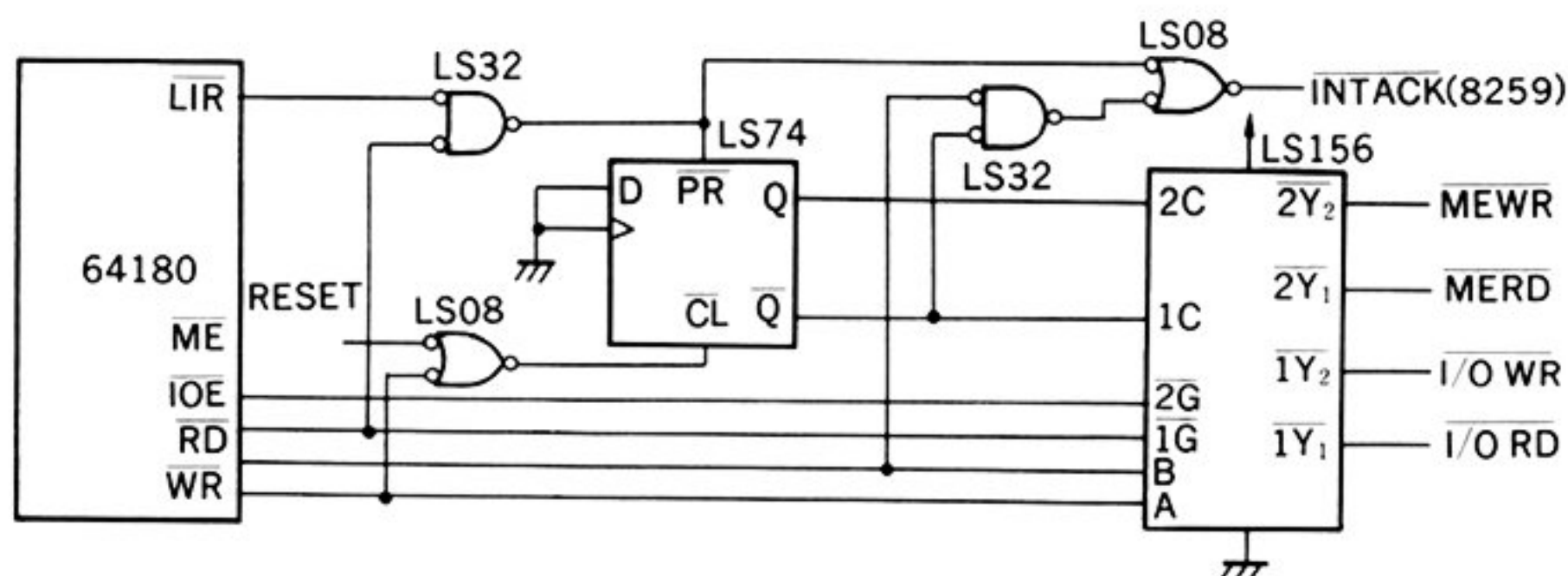



図 9・7 64180 による $\overline{\text{INTACK}}$ 作成例

のタイミングチャートと8259とのインタフェース例を示します.

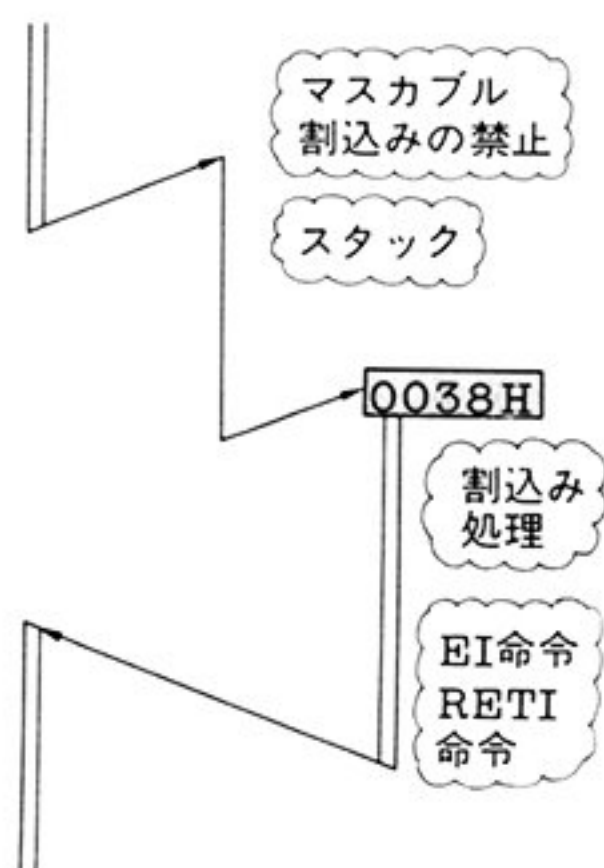
CALL命令のオペコードフェッチ時にはプログラムカウンタ（PC）は停止していますが, Z80とは異なり2バイトのオペランドリード時にはプログラムカウンタが'+2'されています. このため, スタックに退避されるプログラムカウンタの値が実際に戻るべき番地より'+2'されています. したがって, 割込み処理ルーチン上でスタックの値を'−2'する必要があります.

なお, $\overline{\text{INT0}}$をモード0で使用するためには, IM0命令で割込みの動作モードをモード0に設定します. またリセット直後は, モード0にイニシャライズされます.

9・5 $\overline{\text{INT0}}$モード1

このモードでは，プログラムは固定番地（0038H 番地）からスタートします．モード1では，モード0やモード2のように命令やベクタをデータバスにのせる必要がありません．このため，ソフトウェア，ハードウェアとも最も単純なもので済ませることができます．ところが，この番地は，RST␣38H 命令とジャンプ先の番地が重なっているため，もしRST命令（RST␣38H 命令）を使用する場合は注意が必要です．また複数の割込み要因がある場合，モード0，モード2のように拡張性がないという不便さもあります．したがって，小規模なシステムでは大変に便利なモードですが，システム規模が大きく，外部割込み要因が多い場合はモード0かモード2を使う必要があります（**図9・8**）．

図 9・8　$\overline{\text{INT0}}$モード1のシーケンス

$\overline{\text{INT0}}$ をモード1で使用するためには，IM1命令を実行しモード1を設定する必要があります．

EI命令とRETI命令

マスカブルな割込み（$\overline{\text{INT0}}$ など）は，各命令の最後のマシンサイクル中にサンプリングされます．ただしEI 命令の実行中には，割込みは受け付けられません．したがってEI 命令を使用した場合，EI 命令の次の命令から割込みが受け付けられます．

通常EI 命令は，割込み処理の最後にRETI 命令とペアで使用されます．したがって，EI 命令，RETI 命令の順番にプログラムを配置するとRETI 命令は必ず実行され，割込みの要求があった場合は，RETI 命令終了後に割込みアクノレッジサイクルが実行されます．

9・6 $\overline{\text{INT0}}$モード2

〔1〕 **モード2の機能**　このモードは，ザイログ社のZ80周辺LSIなどがサポートしているベクタ方式の割込みです．このモードでは，割込みアクノレッジサイクル中にデータバスから下位側8ビットのベクタアドレスを読み込みます．このベクタアドレスとIレジスタ（上位側8ビットのベクタアドレスとして使用される）の値をあわせて，16ビットのベクタアドレスを生成します．このベクタアドレスから，実際の割込み処理の開始アドレスをリードします．割込み処理は，ここで読み込んだアドレスから開始されます．

したがって，周辺ごとにこの下位側8ビットのアドレスを変更することにより複数（$2^8/2=128$通り）の割込み要因を区別し，独立に処理をさせることができます．

$\overline{\text{INT0}}$をモード2で使用するためには，IM2命令を実行させる必要があります（図9・9）．

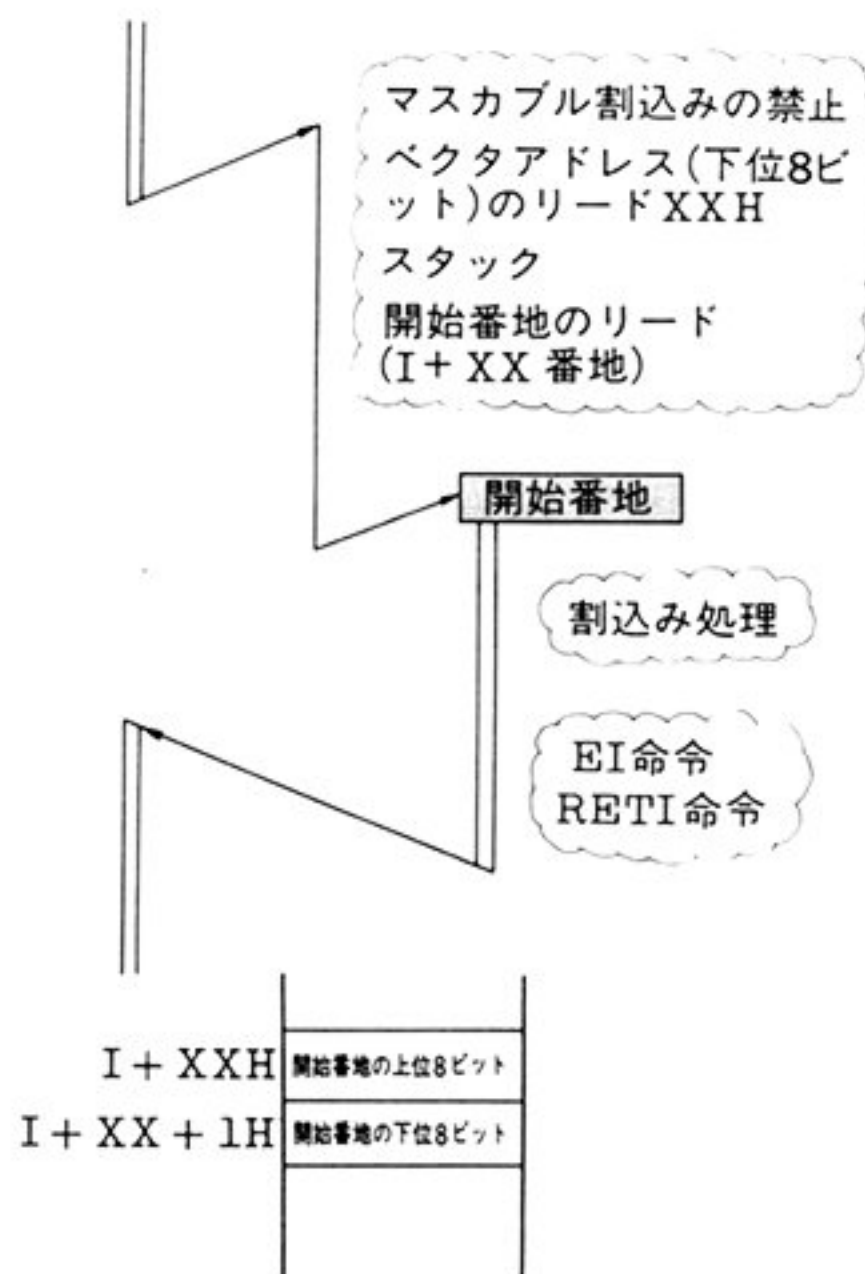



図9・9 $\overline{\text{INT0}}$モード2のシーケンス

〔2〕 **デイジーチェイン**　このモード2では，すでに述べたように複数個の割込み要因をもつことができます．ところが，この状態では異なる要因から同時に要求が入った場合，優先度をつけることができません．そこで，優先順位をつける手法としてZ80周辺LSIを使用した場合，**デイジーチェイン**と呼ばれる方法がとられます．デイジーチェインはハード的に割込みの優先順位を決定する手法で，割込み許可信号（IE信号）により割込みの発生を許可するものです．**図9・10**にデイジーチェインの構成例を示します．このIE信号には入力信号（IEI）と出力信号（IEO）があります．IEI信号は通常‘H’が入力され，‘H’が入力されていればIEOからは‘H’が出力されています．これが割込みの許可されている状態です．この状態で割込みが発生すると，その要因はIEOが‘L’となります．このIEIに‘H’が入りIEOに‘L’が出力されている状態が，割込み処理中の要因です．また，IEIに‘L’が入力されるとIEOも‘L’出力となります．これが割込み発生の禁止された状態です．この機能により，デイジーチェインの高位におかれた割込み要因に対し，優先的に割込み処理を行わせることができます．

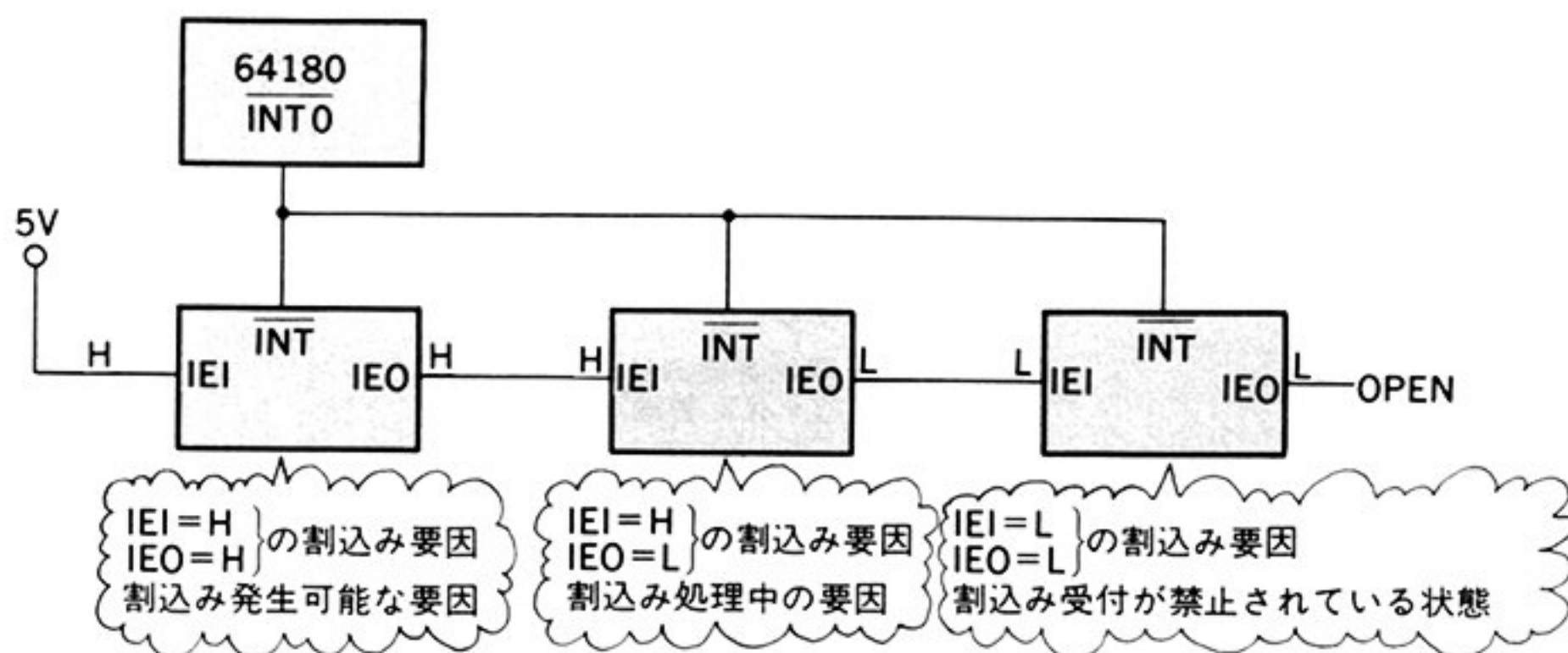



図 9・10　デイジーチェイン

9・7 ノンマスカブルな割込み

〔1〕 $\overline{\text{NMI}}$　$\overline{\text{NMI}}$は，ソフトウェアでマスク不可能な割込みです．$\overline{\text{NMI}}$が発生すると，IEF フラグの値にかかわらず66番へのジャンプが発生します．$\overline{\text{NMI}}$は命令ごとにサンプリングされています．したがって$\overline{\text{NMI}}$が入力されたとき，実行中の命令終了後，$\overline{\text{NMI}}$のアクノレッジサイクルに続き66H番地へのジャンプが発生します．$\overline{\text{NMI}}$は，各命令の最後のマシンサイクルの後から2番目のϕクロックの立下りエッジでサンプリングされています．このタイミングまでに$\overline{\text{NMI}}$が検出されると，そのマシンサイクル終了後，64180は$\overline{\text{NMI}}$のアクノレッジサイクルを実行します．また，$\overline{\text{NMI}}$はエッジセンスであり，一度検出された$\overline{\text{NMI}}$の立下りエッジは，サンプリングのタイミングまで内部に保持されています．この$\overline{\text{NMI}}$処理からの復帰にはRETN命令を使用します（図9・11，図9・12）．

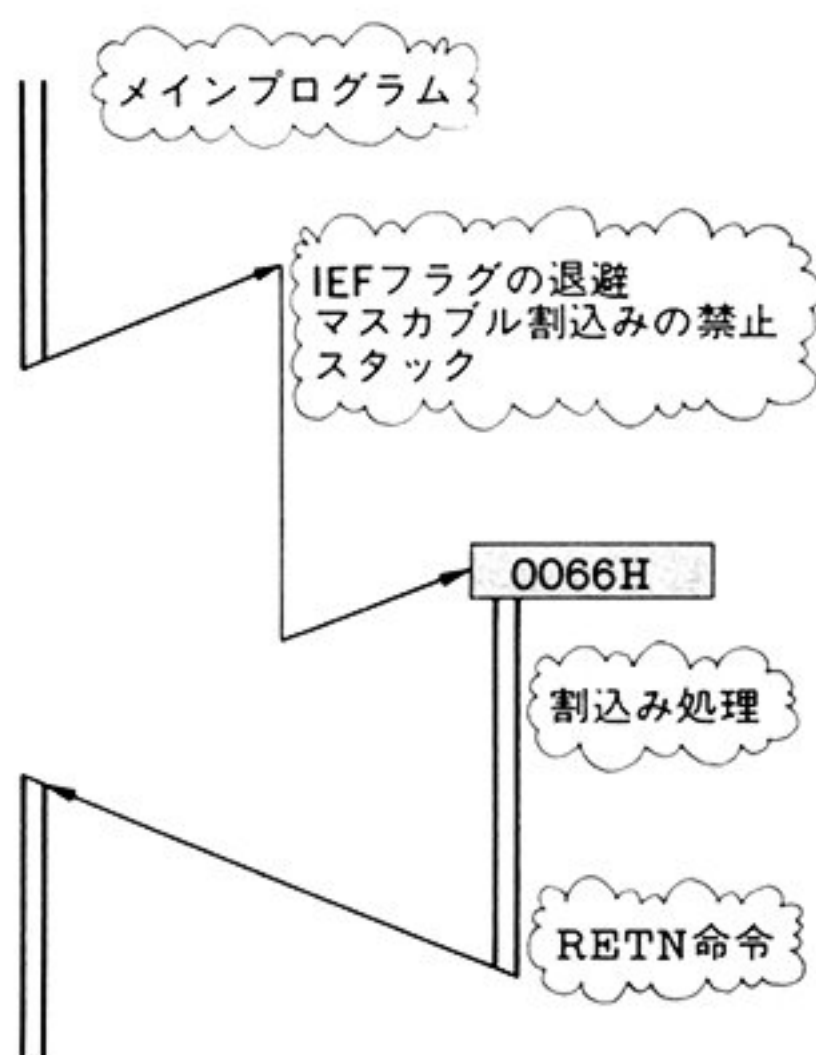
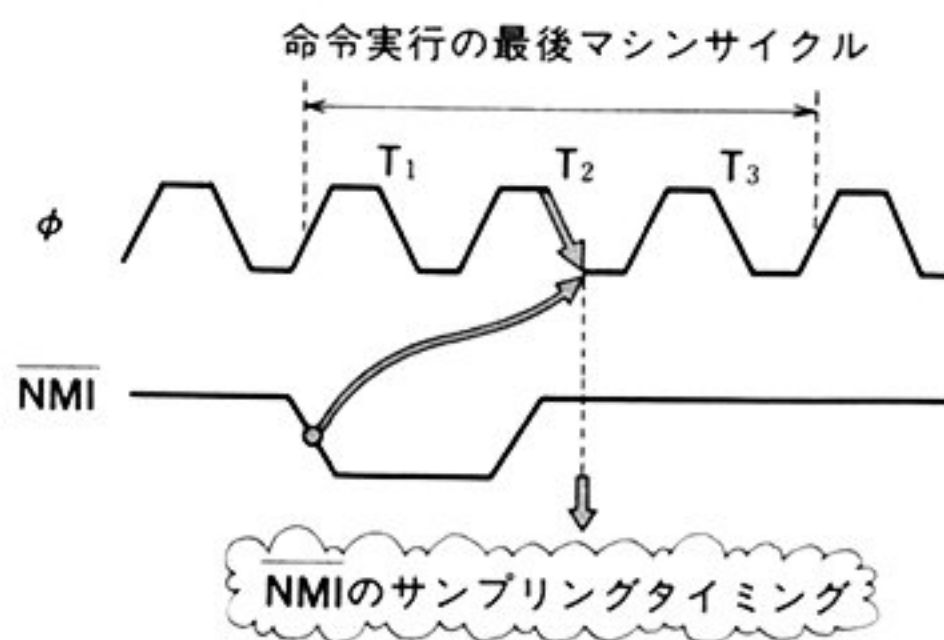



図 9・11　$\overline{\text{NMI}}$のシーケンサ　　図 9・12　$\overline{\text{NMI}}$のサンプリング

〔2〕 $\overline{\text{NMI}}$の用途　通常のプログラム実行に$\overline{\text{NMI}}$が用いられることは少なく，停電などによる電源の遮断などの緊急の用途に使用するのが通例です．また64180の場合，$\overline{\text{NMI}}$が内蔵DMACを一時停止する機能があります．たとえば，DMACがメモリ間のバースト転送を行っている場合，外部I/Oの処理を優先的に行いたい場合，$\overline{\text{NMI}}$を用いてDMACを停止させることができます．

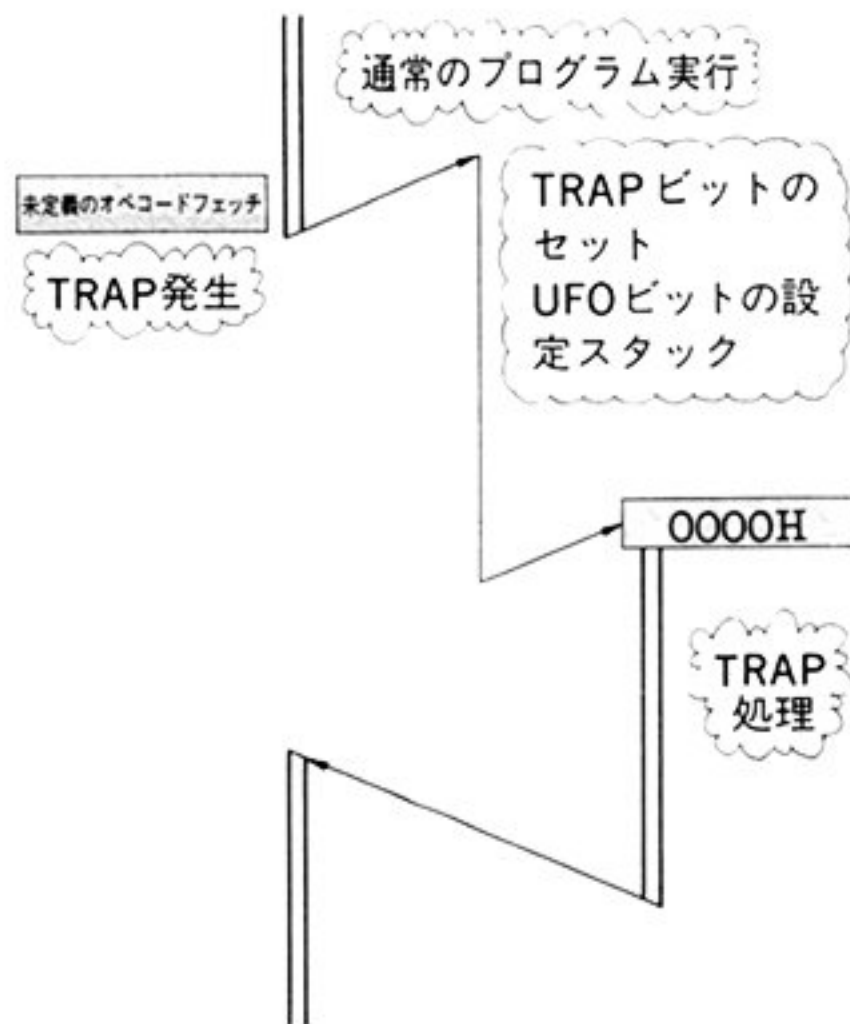



図 9・13　TRAP 割込みのシーケンス

〔3〕 **T R A P**　TRAP はシステムのフェールセーフのための機能です．CPU が未定義のオペコードをフェッチすると割込みが発生します．すなわち，データバス上にのったノイズによるデータの変化や暴走が発生した場合に，システムのリスタートを図るための機能です．未定義コードは，2 バイト，3 バイト目に存在します（**図 9・13**）．

〔4〕 **TRAP の 機 能**　TRAP が発生すると，リセットと同じ 0 番地へのジャンプが発生します．リセットルーチン上で，I/O レジスタにある TRAP ビット（INT/TRAP コントロールレジスタのビット 7）をテストすることにより，0 番地からの実行がリセットか TRAP かを判定することができます．また，同レジスタビットにある UFO ビットを判定することにより，TRAP が 2 バイト目のオペコードで発生したか 3 バイト目のオペコードで発生したかを判定することができます．TRAP はソフトでマスク不可能であるため，TRAP 処理中に再度 TRAP が発生する場合もあります．

9・8 RET命令, RETI命令, RETN命令

〔1〕 **割込みルーチンからの復帰** マスカブル割込みの割込み処理ルーチンからメインルーチンに復帰する場合, 通常RETI命令やRET命令を使用します. この2つの命令の動作は全く同一です. ただし, Z80系の周辺LSIを使用する場合は, RETI命令を使用します. これは, Z80周辺LSIが割込み要求フラグをRETI命令を用いてリセットする(実際にはRETI命令のオペコードをデコードする)ためです(**表9・3**).

表9・3 RET命令, RETI命令, RETN命令

命令	機能	用途
RET命令	スタックの回復 (SP)→PC_L (SP+1)→PC_H SP+2→SP	内部割込み処理ルーチンからの回復サブルーチンからの回復
RETI命令		Z80周辺LSIの割込み処理ルーチンからの回復
RETN命令	スタックの回復 (SP)→PC_L (SP+1)→PC_H SP+2→SP IEFフラグの回復 IEF_2→IEF_1	$\overline{NMI}$処理ルーチンからの回復

〔2〕 **内部割込みとZ80周辺LSI** 内部割込みとZ80周辺LSIを同時に使用する場合は, RET命令とRETI命令を使い分ける必要があります. たとえば, 内蔵の周辺I/Oから割込み処理の終了時にRETI命令を使用すると, 割込み処理が行われていないにもかかわらず外部のZ80周辺LSIの割込み要求がリセットされてしまう可能性があります. これを避けるため内蔵の周辺I/OとZ80周辺LSIを同時に使用する場合, 内蔵の周辺I/Oの割込み処理ルーチンにはRET命令を, Z80周辺LSIの処理ルーチンにはRETI命令を使用します.

〔3〕 **RETN命令とIEFフラグ** $\overline{NMI}$の処理ルーチンからの復帰にはRETN命令を使用します. RET命令やRETI命令との大きな違いはIEFフラグのスタック動作です. $\overline{NMI}$では, IEF1フラグの内容がIEF2フラグに退避されます. このときIEF1フラグは0にリセットされ, マスカブルな割込みは受け付けられま

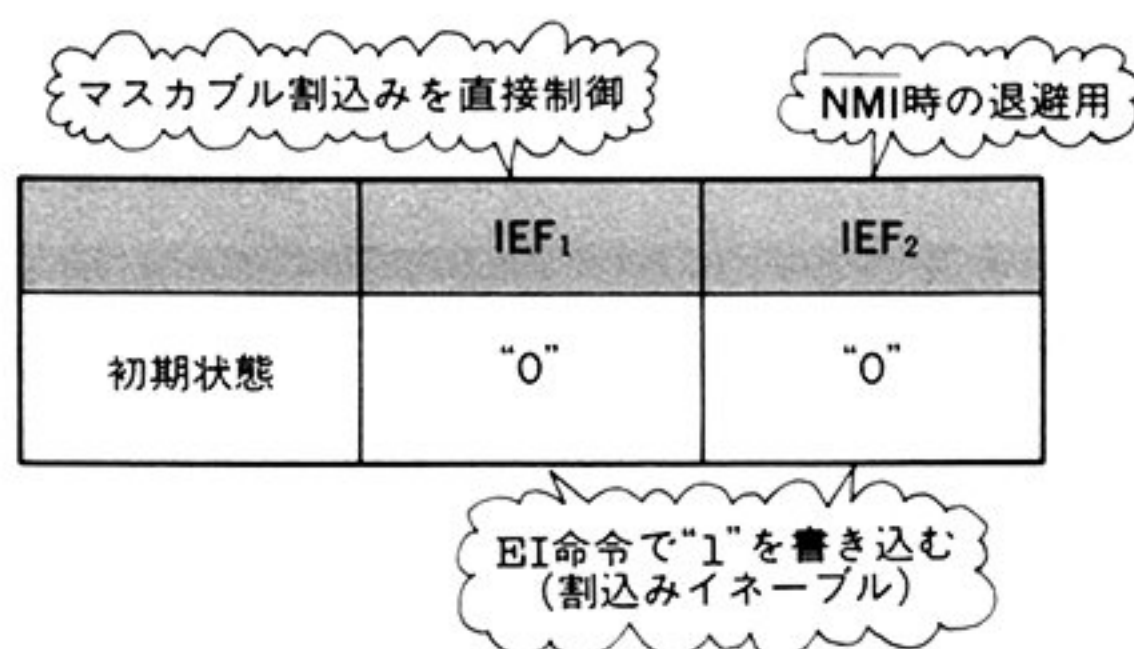



図 9・14　IEF フラグ

せん．RETN 命令ではこれを復帰し，IEF2の内容をIEF1フラグに転送します．これに対し，RET 命令やRETI 命令を実行しても，IEFフラグにはなんの影響も及ぼしません（**図 9･14**）．

このように，内蔵の周辺の割込みにはRET 命令を，Z80 周辺LSIの割込みにはRETI 命令を、$\overline{NMI}$ にはRETN 命令を使用します．

10. 特殊動作モード

64180は，ホルト，バスリリース，低消費電力の３つの特殊動作モードをもっています．ホルトモードは，CPUが処理すべきタスクがなくなり割込み待ちとなるモードです．バスリリースモードは，外部のバスマスタがバスを使用できるように，CPUがバスを解放するモードです．低消費電力モードは，CPU，周辺機能が停止し，消費電力の低減を図るモードです．また，64180を初期化するためリセットの機能があります．

10・1 リセット

パワーオン時，あるいは動作中のシステムを初期化するため，リセットを行います．64180は，リセット端子を6ステート以上アクティブ（Low状態）にすることによりリセット状態になります．リセットにより，CPUと周辺機能は以下に述べるとおりに初期化されます．

〔1〕 C P U　64180は，リセットによりその動作を中止します．リセット信号がアクティブ（Low状態）の間，アドレスバスとデータバスはハイインピーダンス状態になります．$\overline{\text{ME}}$，$\overline{\text{IOE}}$などの制御信号は，インアクティブな状態（'High'）に固定されます（図10•1）．

リセット信号がインアクティブな状態（'High'）になると，CPUは論理アドレスの0000H番地から実行を開始します．リセット直後，MMUのベースレジスタは00Hに初期化されます．このため，見かけ上MMUの動作がバイパスされ，Z80と同様な64Kバイトのアドレス空間をもつCPUに見えます．このとき物理アドレス上では，00000H番地からの実行となります．

図10・1 リセット中の制御信号

図 10・2 リセットのシーケンス

CPU のレジスタのうち，アキュムレータ，汎用レジスタ（B，C，D，E，H，L），インデックスレジスタ（IX，IY）は初期化されません．これに対して，スタックポインタ（SP），I レジスタ，リフレッシュカウンタ（'R'）は，'0' に初期化されます（**図 10・2**）．

また，割込みは IEF フラグが '0' に初期化されるため，マスカブルな割込みは禁止されています．マスカブルな割込みを使用可能とするには，EI 命令を実行する必要があります．

〔2〕 **周 辺 機 能**　リセット直後，64180 の内蔵する DMAC，タイマ，CSI/O，ASCI の動作はディスエーブルされています．これらの周辺機能は，動作を許可するコントロールビットを I/O 空間に配置されたコントロールレジスタ上にもっており，リセット後，これらのビットはすべてディスエーブルされています．リセット直後の各コントロールレジスタの状態は付録を参照して下さい．

10・2 ホルトモード

ホルトモードは，CPU が HALT 命令（76H）を実行した状態です．このモードでは，HALT 命令の次のアドレスに配置された命令を繰り返しフェッチし，プログラムの実行が停止した状態になっています．プログラムカウンタも変化しません．ホルトモード中は，$\overline{\text{HALT}}$ 信号が‘L’となりホルトモードであることを

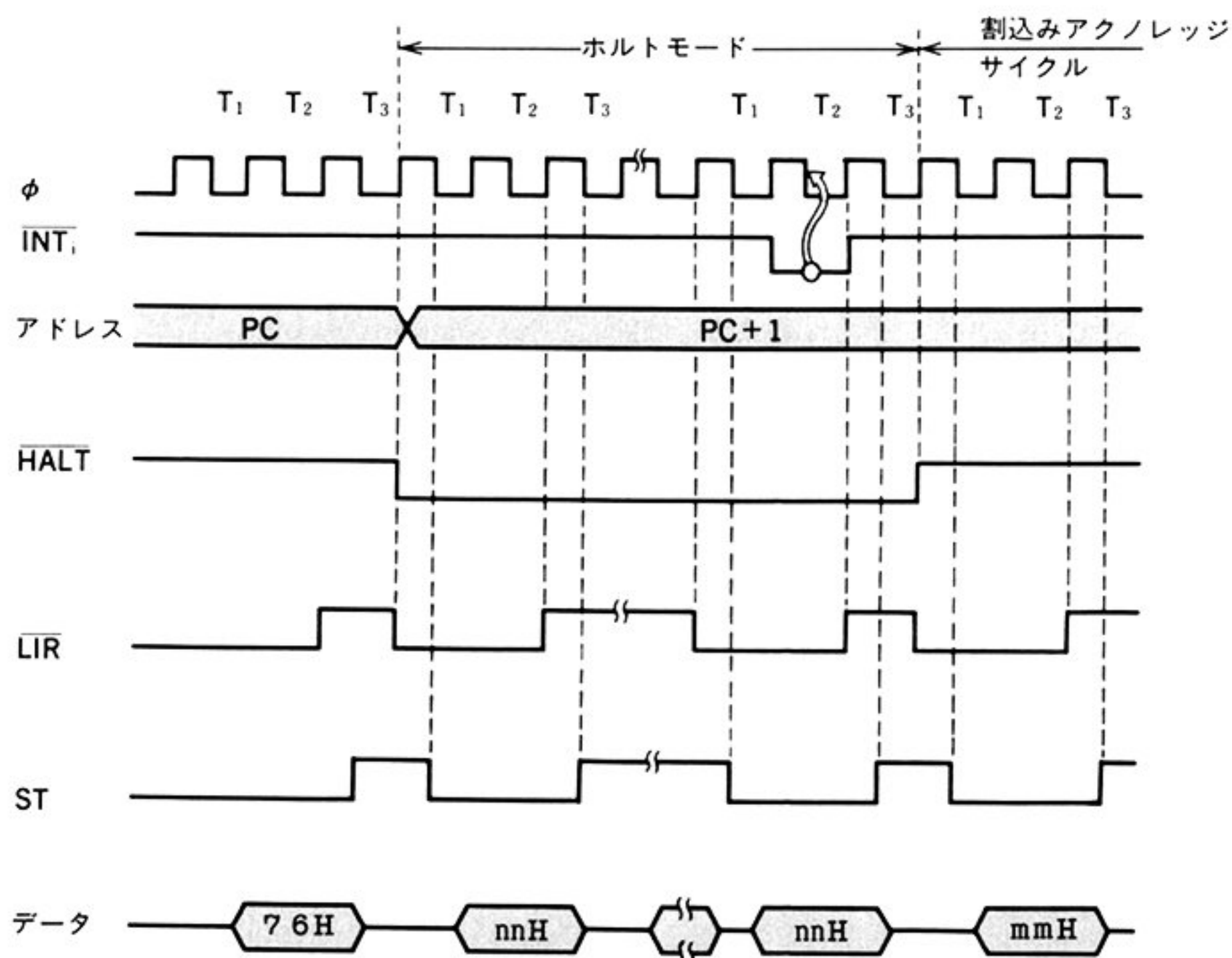



図 10・3 ホルトモード

表 10・1 ホルトモード中の各機能の動作状態

機　　能	状　　態	備　　考
CPU	停　止	割込み，$\overline{\text{BUSREQ}}$ は受付可
リフレッシュ	継続動作可能	—
DMAC	継続動作可能	—
CSI/O	継続動作可能	—
ASCI	継続動作可能	—
タイマ	継続動作可能	—
CPG	継続動作	—

知らせます．また，命令を繰り返しフェッチしていますから，$\overline{\text{LIR}}$ 信号が繰り返し出力されます（**図 10・3**）．

ホルトモード中でも，リフレッシュコントローラや DMAC，および他の周辺機能（タイマ，CSI/O，ASCI）は継続して動作します．また，割込みや $\overline{\text{BUSREQ}}$ 要求も受付可能です（**表 10・1**）．

ホルトモードから通常動作モードに復帰するためには，割込みかリセットを使用します．割込みの要因としては，内部，外部の要因を使用可能です．ただし，$\overline{\text{NMI}}$ を除いた他のマスカブル割込みを使用する場合，IEF フラグが '1' にセットされていないと，ホルトモードは解除されません．

ホルトモードはシングルタスクの応用で，I/O デバイスの処理を実行中，他に処理すべきタスクがない場合，I/O の処理の終了をもつ場合などに使用します．

ホルトモードとスリープモード

ホルトモードもスリープモードも，機能的には CPU が停止し割込み待ちの状態になり，同一のものといえます．スリープモードの場合，ホルトモードが命令のフェッチを行っているのに対し，命令フェッチそのものも停止し低消費電力化を図っています．したがって，消費電力が問題となる場合はスリープモードを使用するとよいでしょう．一方，スリープモードの場合リフレッシュが行われなくなるため，DRAM を使用する場合はホルトモードを使用するとよいでしょう．

10・3 バスリリースモード

$\overline{\text{BUSREQ}}$ 端子をアクティブ（Low 状態）にすることにより，64180 は**バスリリースモード**に入ります．バスリリースモードは，マルチ CPU システムや外部の DMAC など，外部のバスマスタとなり得る LSI に対しバス権を与えるための機能です．$\overline{\text{BUSREQ}}$ 信号は，通常動作モードの場合，T_3 サイクルの 1 つ前の ϕ クロックの立下りでサンプリングされます．また，バスリリースモード中は毎サイクルサンプリングされており，サンプリングされたクロックの次の ϕ クロックでバスリリースモードが終了し，通常動作モードとなり 64180 の CPU または DMAC がバスマスタとなります．**図 10・4** に $\overline{\text{BUSREQ}}$ 信号のサンプリングタイミングを示します．バスリリースモードに入ると，アドレスバス，データバス，$\overline{\text{ME}}$，$\overline{\text{IOE}}$，$\overline{\text{RD}}$，$\overline{\text{WR}}$ 信号がハイインピーダンス状態になり，外部のデバイスがバスマスタとなることができます．また，バスリリースモード中は $\overline{\text{BUSACK}}$ 信号がアクティブ（Low 状態）となり，外部のバスマスタに対し 64180 のバスがハイインピーダンスとなり，システムバスを使用可能であることを知らせます．

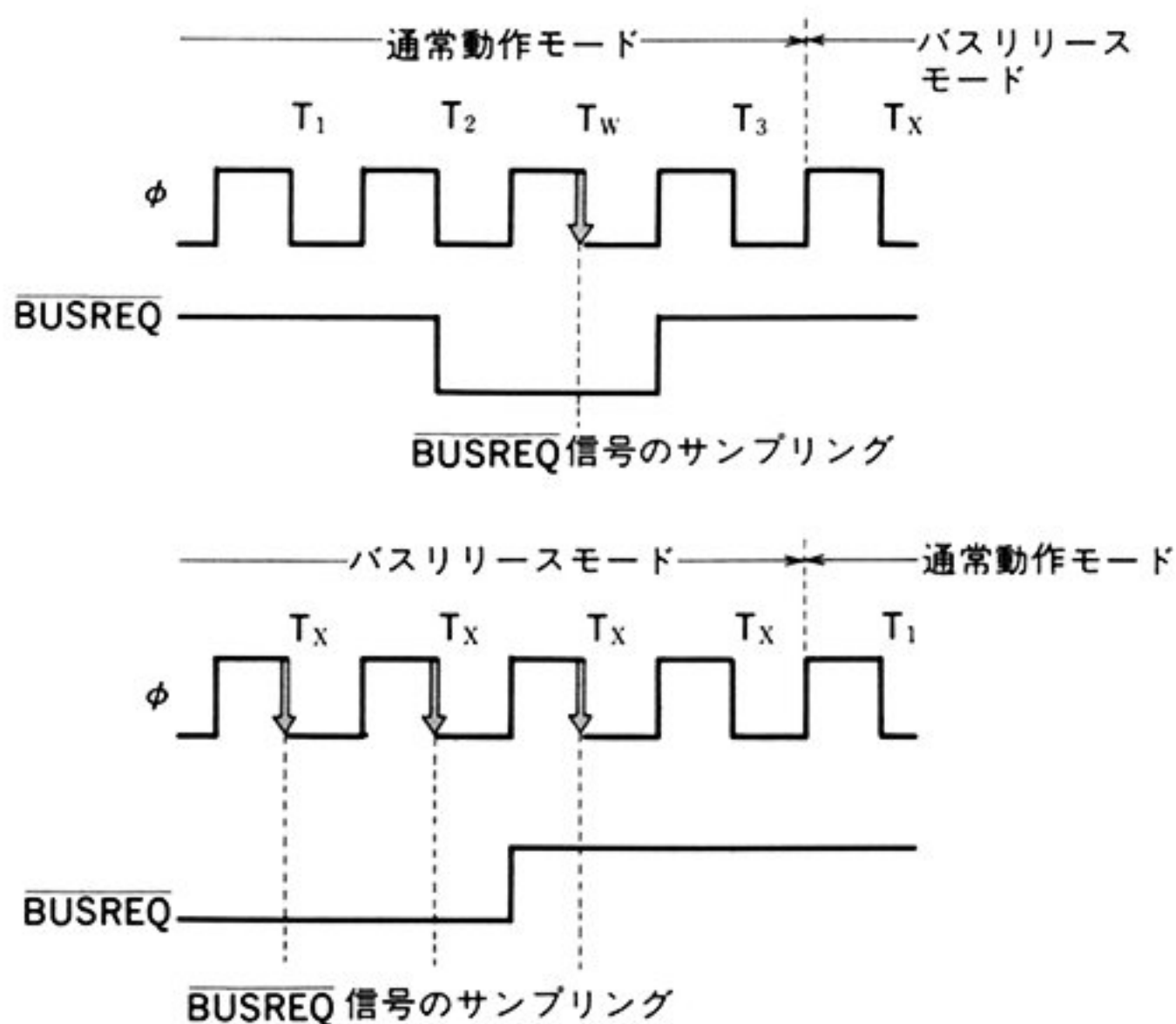



図 10・4 $\overline{\text{BUSREQ}}$ 信号のサンプリング

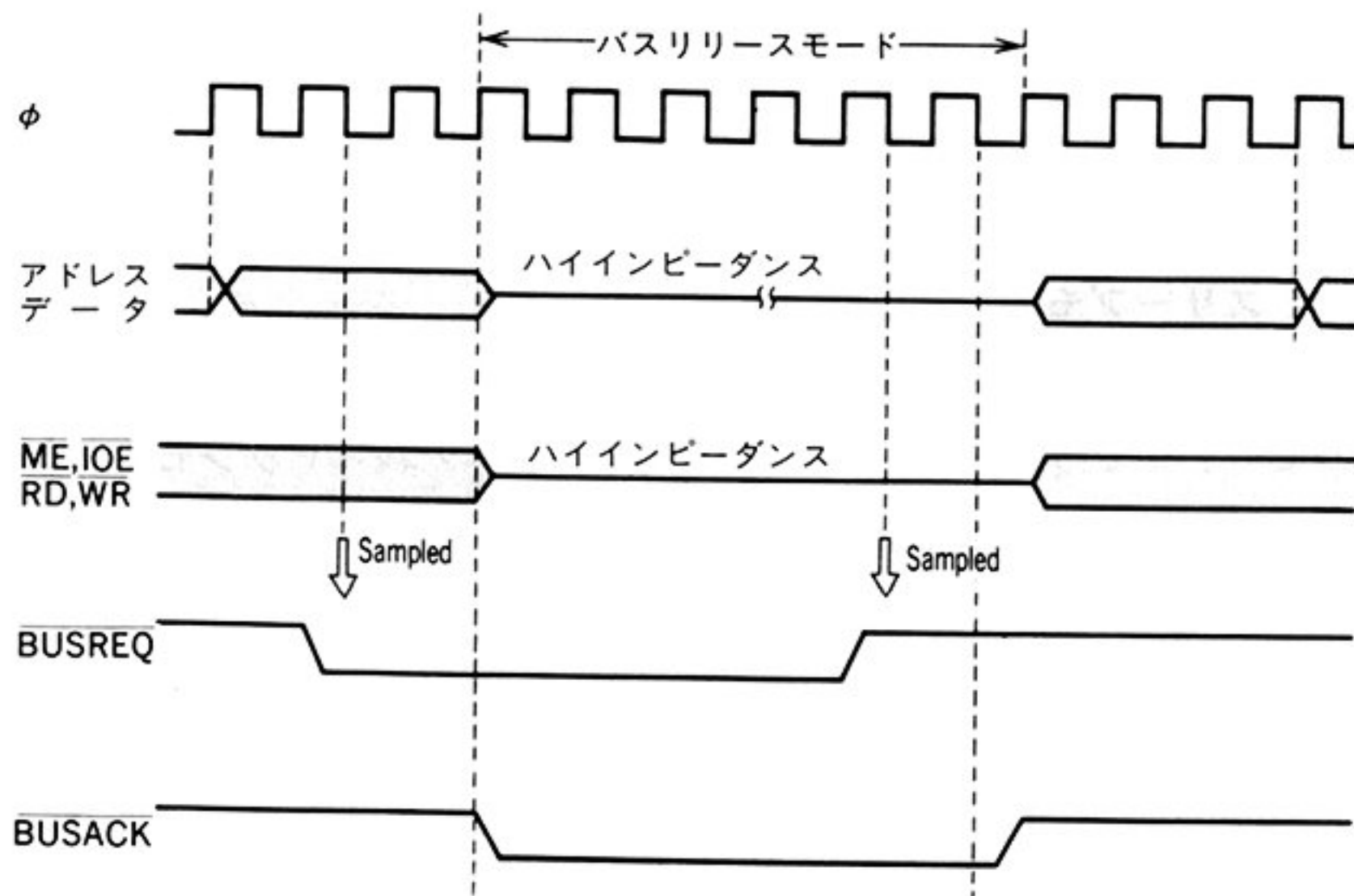



図 10・5 バスリリースモード

バスリリースモード中は割込みは受け付けられず，リフレッシュサイクルは挿入されません．したがってバスリリースモード中は，バスマスタとなった外部のデバイスがDRAMのリフレッシュを行う必要があります．このときでも，周辺機能（タイマ，CSI/O，ASCI）は継続して動作します（**図10･5**）．

また，内蔵のDMAが動作中でも$\overline{\text{BUSREQ}}$信号は受け付けられます．このため，外部のDMACが優先的にバスを使用することができます．

バスリリースモードから通常モードに戻るためには，前述のとおり，$\overline{\text{BUSREQ}}$端子をインアクティブな状態（'High'）にします．この手順により，64180のCPUまたはDMAがバス権を獲得し，$\overline{\text{BUSACK}}$信号がインアクティブな状態（'High'）となります．

10・4 低消費電力モード

64180は，低消費電力モードとして

スリープモード

システムストップモード

の2つのモードをもっています．これ以外に，システムストップモードへの遷移状態として，I/Oストップモードと呼ばれる状態があります．

表 10・2 低消費電力モード

	モードの設定	オペレーション		モードの解除
		CPU	周辺	
スリープモード	SLP命令の実行	停止	動作	$\overline{\text{RESET}}$ 割込み
システムストップモード	IOSTPビット（I/Oコントロールレジスタ）をセットした状態でのSLP命令の実行	停止	停止	$\overline{\text{RESET}}$ 外部割込み

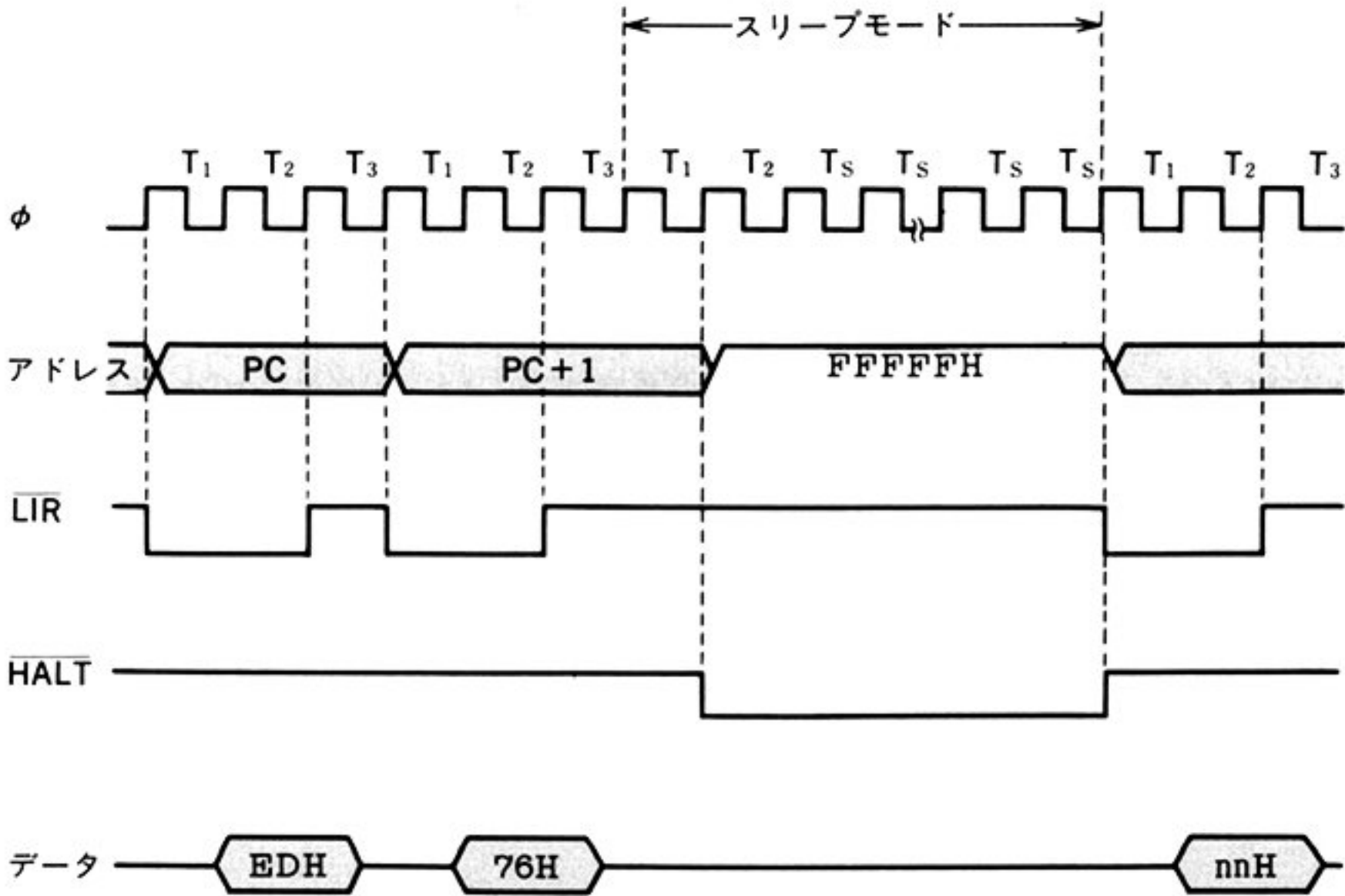



図 10・6 スリープモード

いずれのモードも，発振回路およびクロック出力回路は停止していません．したがって，Eクロックとϕクロックは継続して出力されます（**表10・2**）．

（1） **スリープモード**　CPUがSLP命令（ED 76H）を実行すると，64180はスリープモードに入ります．スリープモードに入ると，CPUは命令の実行を停止し，消費電力は通常動作時の60％程度になります．また，リフレッシュサイクルは挿入されず，DMACも動作しません．このとき，これ以外の周辺機能（タイマ，CSI/O，ASCI）は継続して動作します．$\overline{\text{BUSREQ}}$信号は受け付けられ，外部のLSIがバスマスタになることができます．

スリープモードを解除するには，割込みかリセットを使用します．スリープモードの解除にマスカブルな割込みを使用する場合，IEFフラグが'1'にセットされているときは，通常の割込み処理ルーチンを実行します．これに対し，IEFフラグが'0'にリセットされている場合は，SLP命令の次の命令から実行が再開されます（**図10・6**）．

（2） **システムストップモード**　I/Oコントロールレジスタ（I/O空間の3FH番地に配置）上のIOSTPビット（ビット5）を'1'にセットした状態で，SLP命令を実行することにより，64180はシステムストップモードに入ります．

このモードでは，CPU，周辺機能（DMAC，タイマ，CSI/O，ASCI）も停止した状態となり，消費電力は通常動作時の1/4程度になります．

システムストップモードを解除するためには，外部割込みとリセットを使用します．割込みによりCPUは動作を再開しますが，IOSTPビットは，'1'にセットされた状態で周辺機能は停止しています．周辺機能の動作を再開させるためには，IOSTPビットに'0'を書き込みます．

11. DMA コントローラ

64180は，2チャネルのDMAコントローラを内蔵しています．DMACはCPUに代わり，メモリ↔メモリ，メモリ↔I/Oの高速なデータ転送を行うための周辺機能です．たとえば，フロッピーディスクコントローラやハードディスクコントローラとのデータ転送には欠かせない機能です．DMACはマルチタスクを処理するためには必須の機能で，CPUは一度DMACに起動をかけた後は，データ転送をいっさい気にせず，プログラムの処理を行うことができます．

11・1 DMAC とは

データをI/Oデバイスからメモリに転送したり，あるメモリエリアから他のメモリエリアに転送したりする手法には，大きく分けて次のような3通りの方法があります（**図11・1**，**表11・1**）．

(1) CPU（プログラム）によるデータ転送…ポーリング

(2) 割込みによるデータ転送

(3) DMACによるデータ転送

(1)，(2)はいずれも，データ転送はCPUが行います．(1)は転送すべきデータ

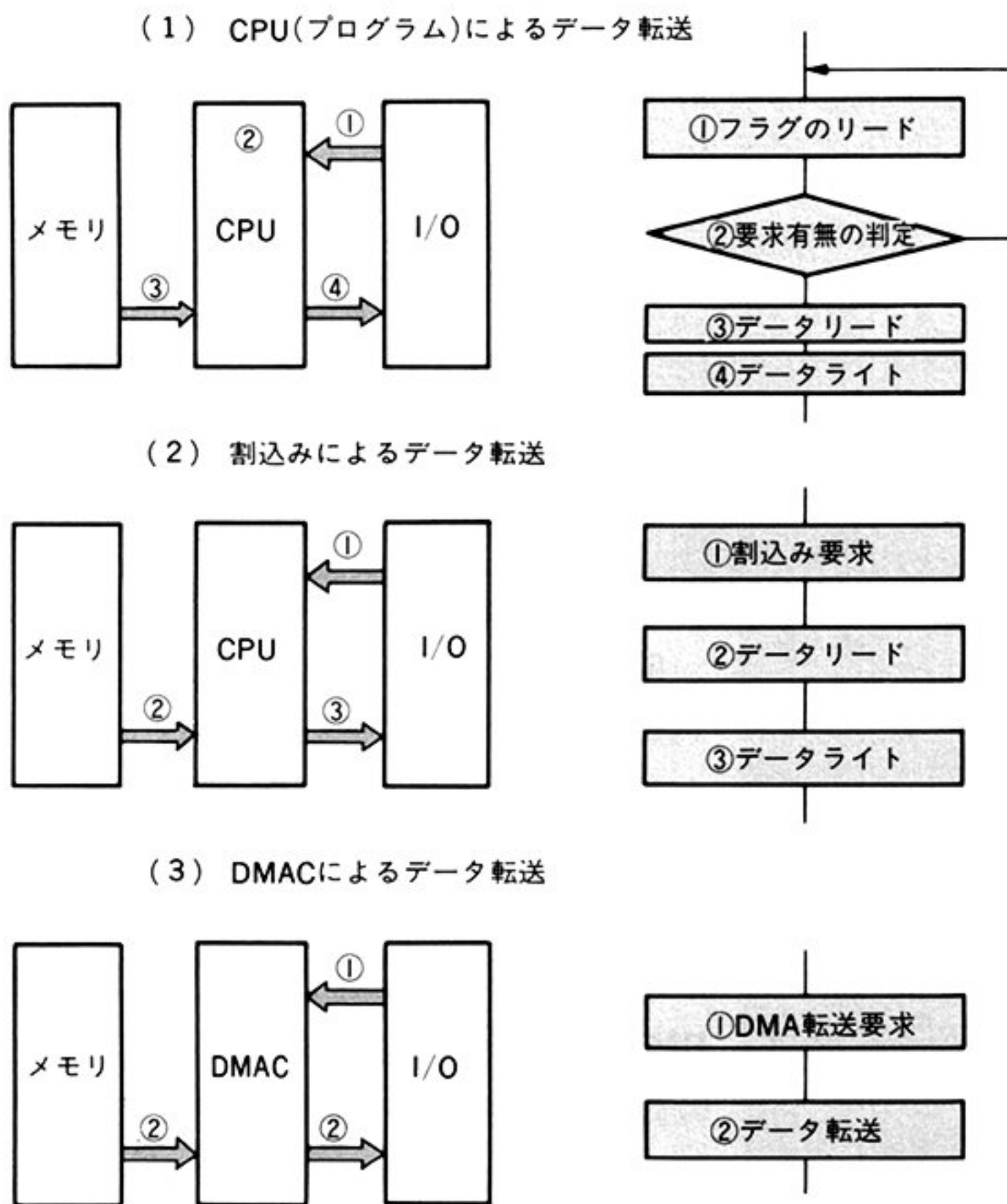



図 11・1 データ転送の手法

表 11・1 データ転送の手法の比較

	長所	短所
プログラムによる転送(ポーリング)	一番手軽なデータ転送	外部との同期がとりにくい プログラム処理速度が遅い
割込み	外部との同期がとりやすい	アクノレッジサイクルによる速度の低下 割込みのハードが必要
DMAC	転送速度が速い 外部との同期がとりやすい	DMA の専用ハードが必要

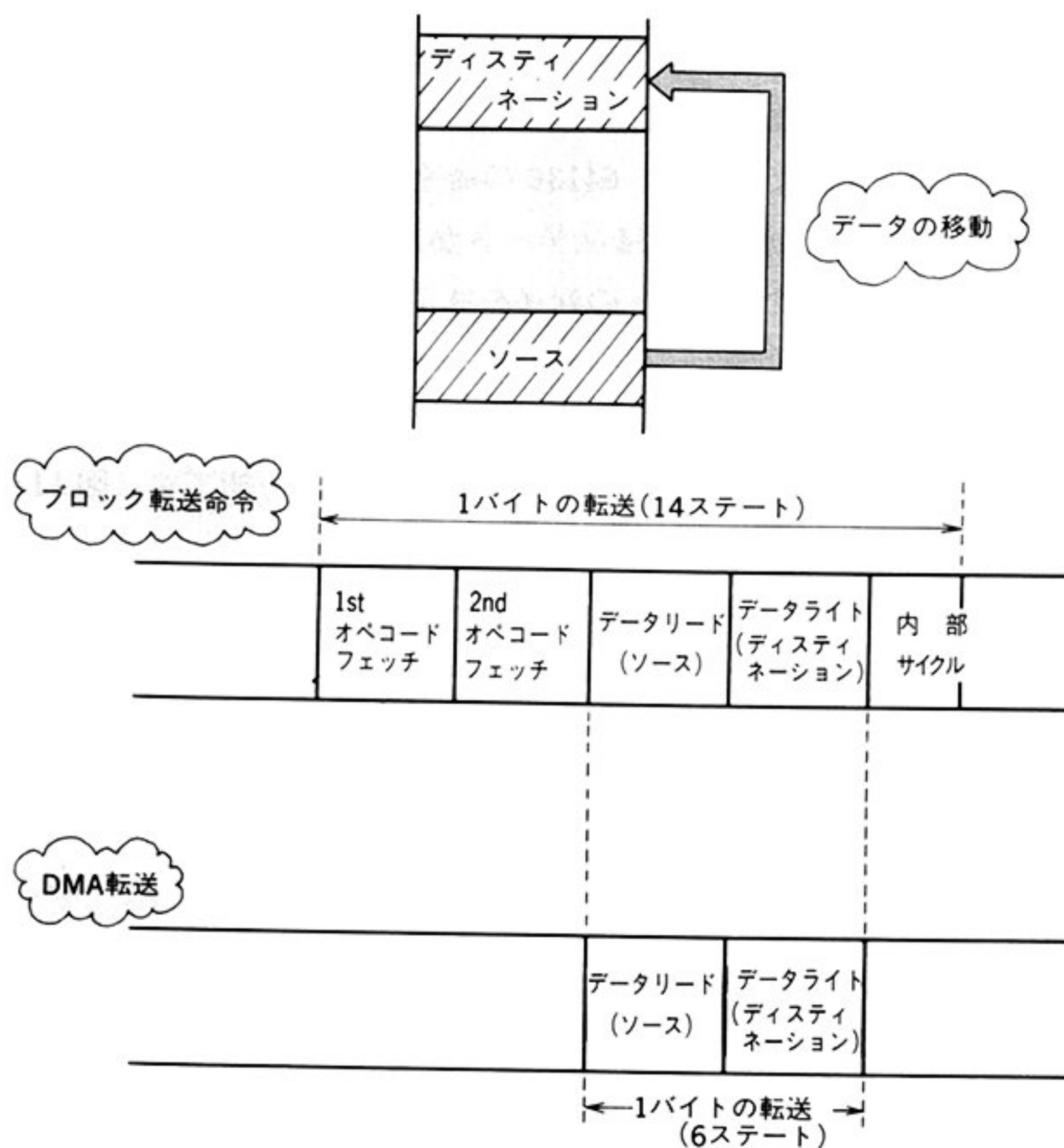



図 11・2 ブロック転送命令と DMA 転送

の有無をCPUがチェックし，命令でデータのリードおよびライトを独立に行う必要があります．また，(1)の手法では，データの有無の判定およびデータ転送を行う間，CPUが他のプログラムを実行できないため，システムのスループットが下がるという欠点があります．

(2)の手法では，データの有無を判定するためのプログラムが，割込みにより省略可能となります．しかし，(1)，(2)の手法はプログラムによりデータ転送を行うため，転送速度には限界があります．

(3)のDMACはCPUからバス権を受け取り，データ転送を専門に行う周辺機能です．このDMACによるデータ転送では，数Mバイト/Sの転送も可能となります．プログラム上では，DMACに起動をかけた後は，CPUはデータ転送に関し全く意識することなくプログラムを実行可能です．

CPUが行うデータ転送との大きな違いは，データ転送のため命令をフェッチする必要がない点です．たとえば，64180の命令でデータ転送に適したブロック転送命令でも，1バイトの転送に14ステートかかります．これに対しDMACを使用すれば，6ステートで1バイトの転送を終了することができます．

64180はこのように，高速データ転送に適したDMACを2チャネル内蔵しています．チャネル0ではメモリ↔メモリ，メモリ↔I/O（メモリマップドI/Oも含む）の転送が，チャネル1ではメモリ↔I/Oの転送が可能です（**図11・2**）．

11・2 DMACの構造

〔1〕 **DMACのレジスタ**　DMACには，大別して，アドレスレジスタ，バイトカウントレジスタ，コントロール/ステイタスレジスタの3つのレジスタが内蔵されています．64180は2チャネルのDMACを内蔵しており，アドレスレジスタとバイトカウントレジスタは，チャネルごとに独立に，またコントロールレジスタを共通にもっています（**図11・3**）．

〔2〕 **アドレスレジスタ：チャネル0**　チャネル0ではメモリ↔メモリ，およびメモリ↔I/Oのデータ転送が可能です．チャネル0はDMA転送を行う，データの転送元を示すソースアドレスレジスタ（20ビット，I/O空間の20H，21H，22H番地に配置），およびデータの転送先を示す．ディスティネーションアドレスレジスタ（20ビット，I/O空間の23H，24H，25H番地に配置）の2種類のアドレスレジスタをもっています．

〔3〕 **アドレスレジスタ：チャネル1**　チャネル1では，メモリ↔I/Oのデータ転送が可能です．チャネル1は，メモリの転送開始アドレスを設定するメモ

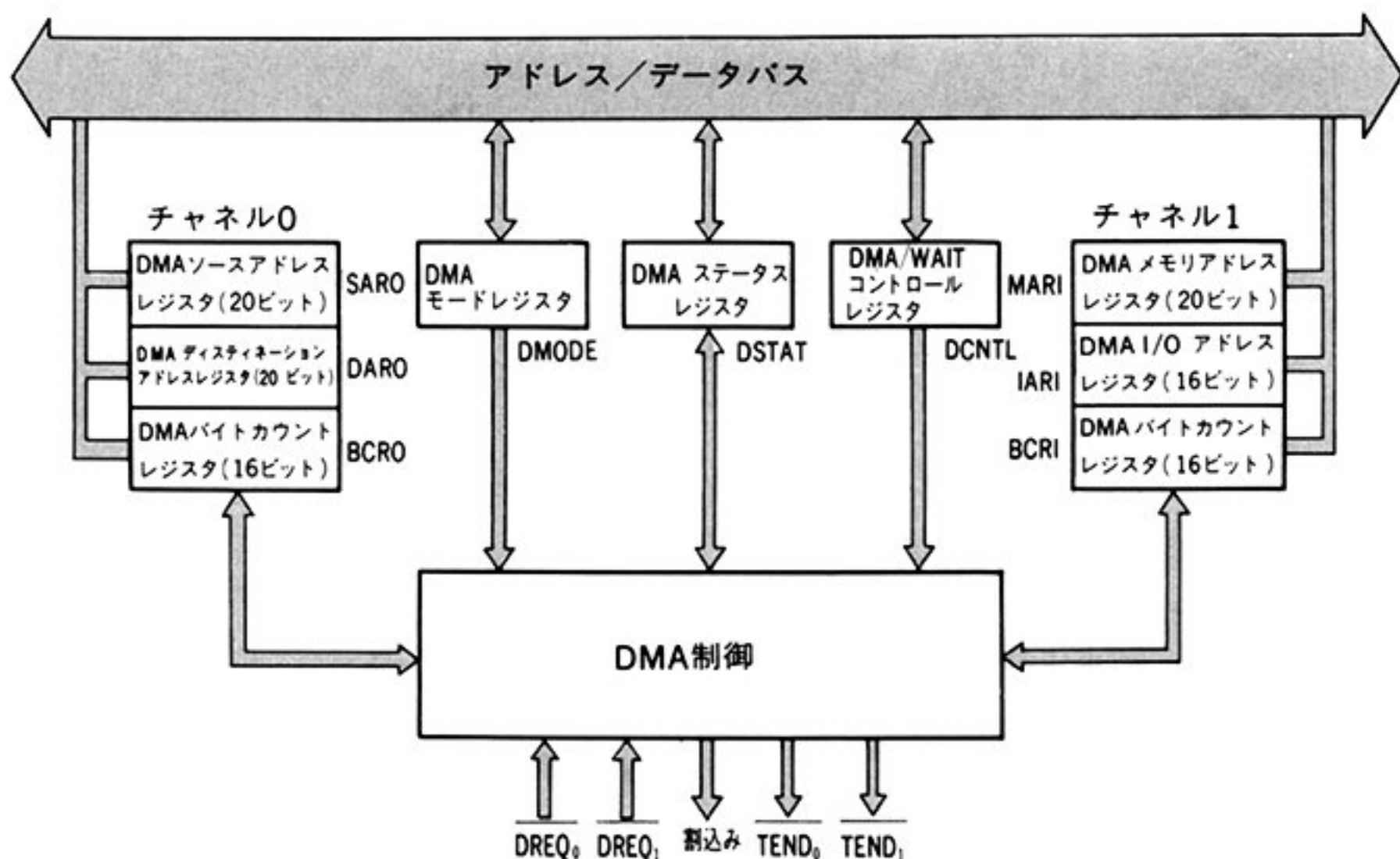



図 11・3　DMACブロック図

リアドレスレジスタ（20ビット，I/O空間の28H，29H，2AH番地に配置）と，I/Oアドレスを設定するI/Oアドレスレジスタ（16ビット，I/O空間の2BH，2CHに配置）をもっています．

〔4〕**バイトカウントレジスタ**　DMA転送を行うデータの数（バイト数）を設定するレジスタです．たとえば，10バイトのデータ転送を行う場合，バイトカウントレジスタには0AHを設定します．また，バイトカウントレジスタに00Hを設定すると，最大の64Kバイトの転送を行います．チャネル0，1とも16ビットのバイトカウントレジスタをもっており，チャネル0のバイトカウントレジスタはI/O空間の26H，27H番地に，チャネル1のバイトカウントレジスタは2EH，2FHに配置されています．

〔5〕**コントロール/ステータスレジスタ**　チャネル0，1の動作モードや割込みを制御するため，3バイトの制御レジスタをもっています．1つは，チャネル0，1の動作のイネーブル，ディスエーブルや割込み制御を行うDMAステータスレジスタ（I/O空間の30Hに配置），1つは，チャネル0のメモリ↔メモリなどの動作モードを設定するためのDMAモードレジスタ（I/O空間の31Hに配置），もう1つは，チャネル1の動作モードを設定するためのDMA/WAITコントロールレジスタ（I/O空間の32Hに配置）の3つのレジスタです．

11・3 DMACの転送モード

〔1〕 **メモリ↔メモリ**　64180の内蔵したDMACのチャネル0では，メモリ↔メモリのDMA転送が可能です．64180のDMACは，デュアルアドレス方式をとっています．デュアルアドレス方式は，従来DMACの主流であった，データのリードとライトを同時に行うシングルアドレス方式に対し，リードとライトが独立に発生する方式です．**表11・2**にシングルアドレス方式とデュアルアドレス方式の比較を示します．したがって，ウェイトステートが挿入されないとすると，最小バスサイクルは6ステート（リード3ステート，ライト3ステート）です．最大転送レートは1.3Mバイト/s（f=8 MHz）です．

表 11・2　シングルアドレス方式とデュアルアドレス方式

	バスサイクル	DACK信号*	備　考
シングルアドレス方式	1サイクル（1マシンサイクル）	必　要	主にメモリ↔I/Oの転送
デュアルアドレス方式	2サイクル（2マシンサイクル）	不　要	シングルアドレス方式に比べると速度は遅い

*DACK信号…DMACから出力されるチップセレクトに相当する信号

転送元，転送先のメモリアドレスの指定は，MMUを介さず1MBの物理アドレス空間のアクセスを行うため，20ビットのアドレスレジスタにより行います．この機能により，論理空間上にない物理空間上のデータを転送することが可能となります．

メモリ↔メモリの転送には，2つの転送モードがあります．1つは，DMA転送を開始すると，バイトカウントレジスタが‘0’になり，転送が終了するまでDMACがバスを専有するバーストモードと，もう1つは，1バイトのDMA転送ごとにCPUの1マシンサイクルを実行し，これを交互に繰り返すサイクルスチールモードです．バーストモードでは，DMACがバスを専有しCPUサイクルは実行されないため，転送が終了しCPUに制御が戻るまで割込みが受け付けられませんが，サイクルスチールモードではCPUとDMACが交互にバスを使用するため，DMA転送を行いながら割込み処理を行うことも可能です（**図11・4**）．

なお，バーストモードでDMACがバスを専有している場合でも，リフレッシュコントローラによりリフレッシュサイクルを挿入可能で，DRAMの内容を保持

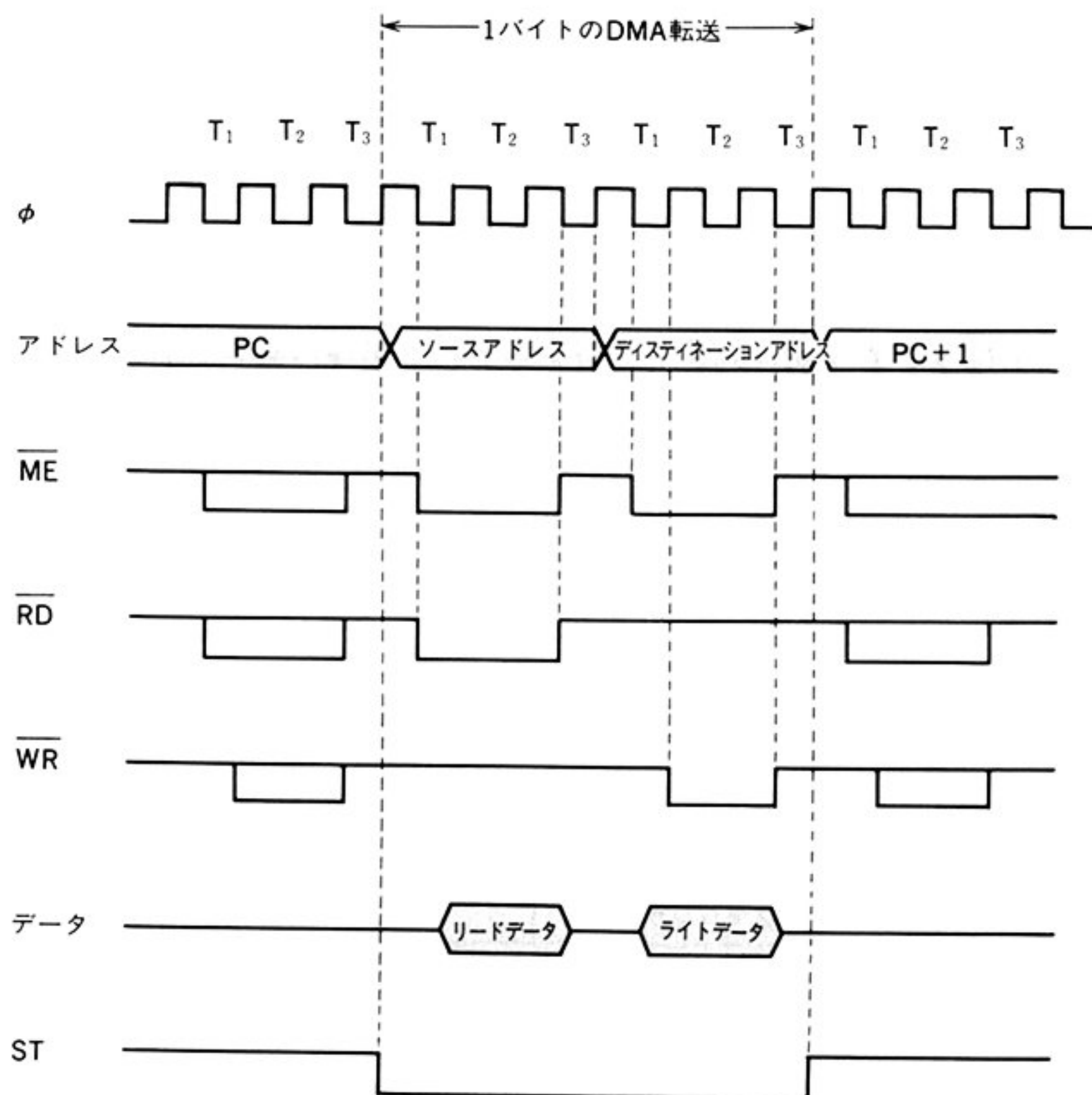



図 11・4 メモリ↔メモリのDMA転送

することができます

〔2〕 **メモリ↔I/O**　チャネル0，1ともメモリ↔I/OのDMA転送が可能です．また，チャネル0ではメモリ↔メモリマップドI/Oの転送も可能です．さらに，チャネル0では内蔵の非同期シリアル（ASCI）とのDMA転送も可能です．メモリ↔I/Oもデュアルアドレス方式をとっているため，メモリとI/Oへのリードおよびライトサイクルが独立に発生します．I/OをDMA転送の対象として選択する場合，CE信号に代わり$\overline{\mathrm{ACK}}$信号（DMA転送の要求信号：$\overline{\mathrm{DREQ}}$信号に対する応答信号）を返す必要があります．デュアルアドレス方式の場合，メモリサイクルと独立に発生するI/Oリード，あるいはI/Oライトサイクルに出力されるI/Oアドレスをデコードし，$\overline{\mathrm{ACK}}$信号として使用します．

メモリ↔I/OのDMA転送を行うためには，DMAの転送をイネーブルにした

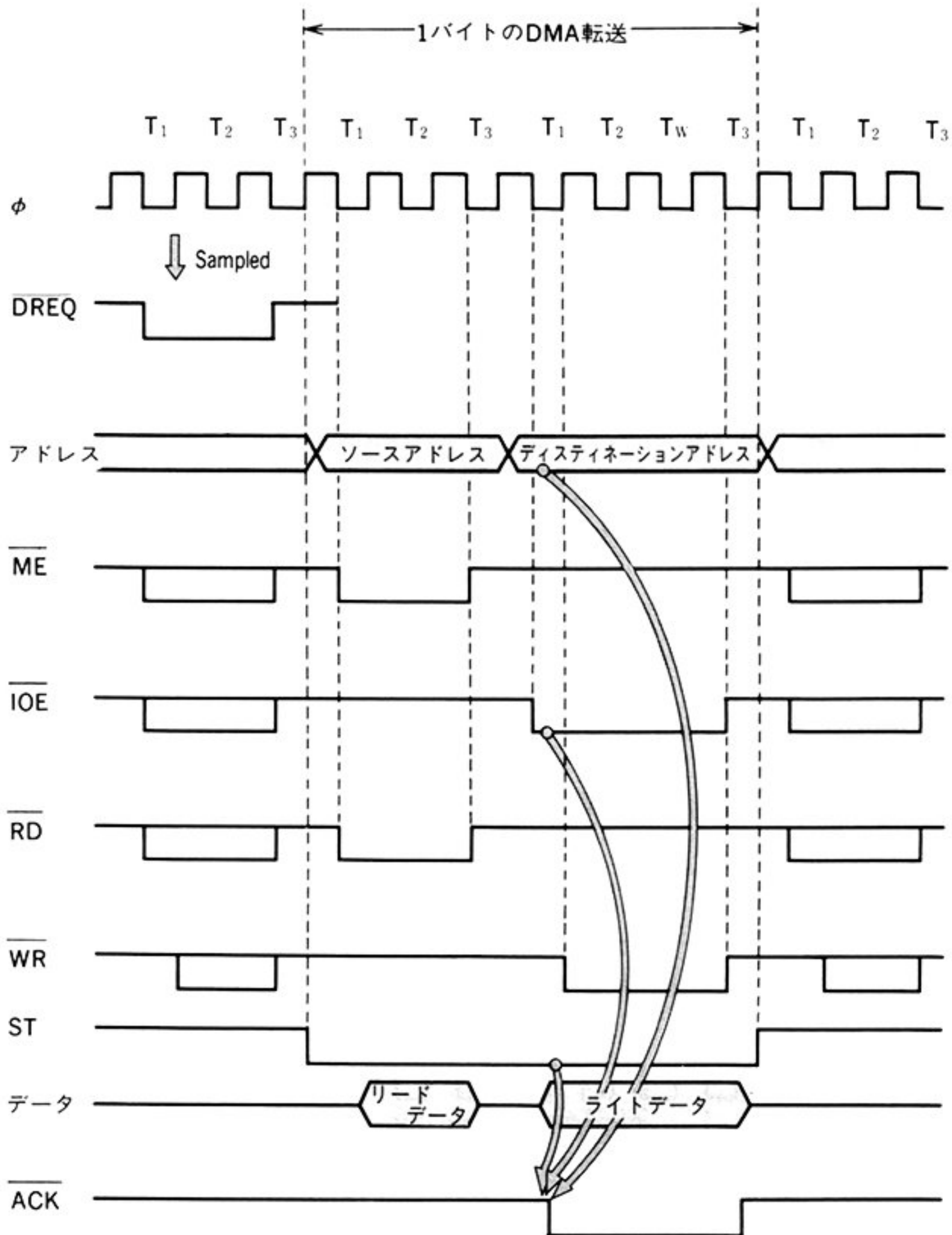



図 11・5 メモリ↔I/OのDMA転送(メモリ→I/Oの例)

状態(DEビットに'1'をセットした状態)で，$\overline{\text{DREQ}}$信号をアクティブ(Low)にします．この$\overline{\text{DREQ}}$信号は，コントロールレジスタの設定により，エッジセンスとレベルセンスを選択可能です．1バイトごとに同期をとって転送を行うにはエッジセンスが，連続して転送を行うにはレベルセンスが適しています(**図11・5**)．

また，レベルセンスの場合要求はラッチされず，$\overline{\text{DREQ}}$信号はサンプリング

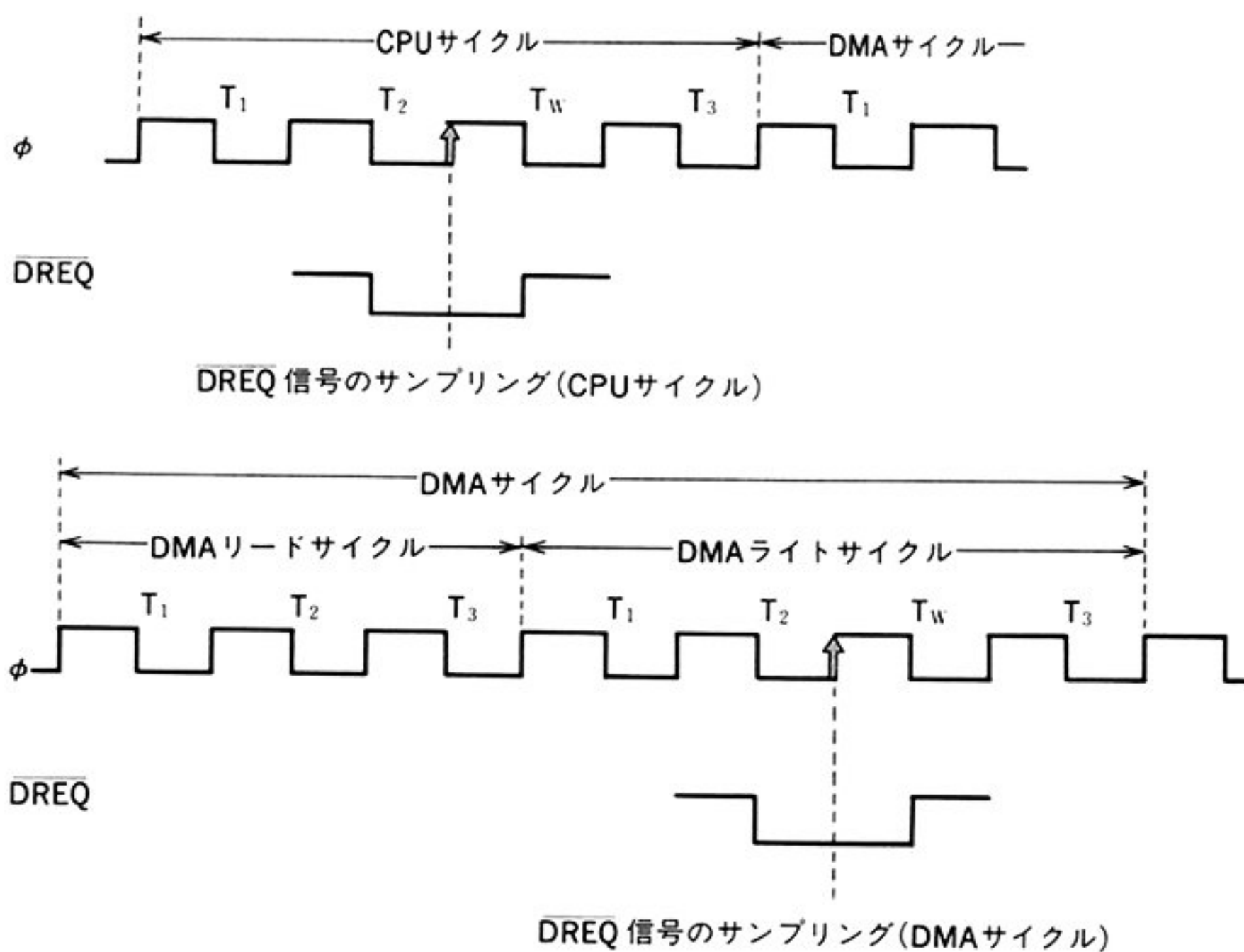



図 11・6 $\overline{\text{DREQ}}$信号のサンプリングタイミング

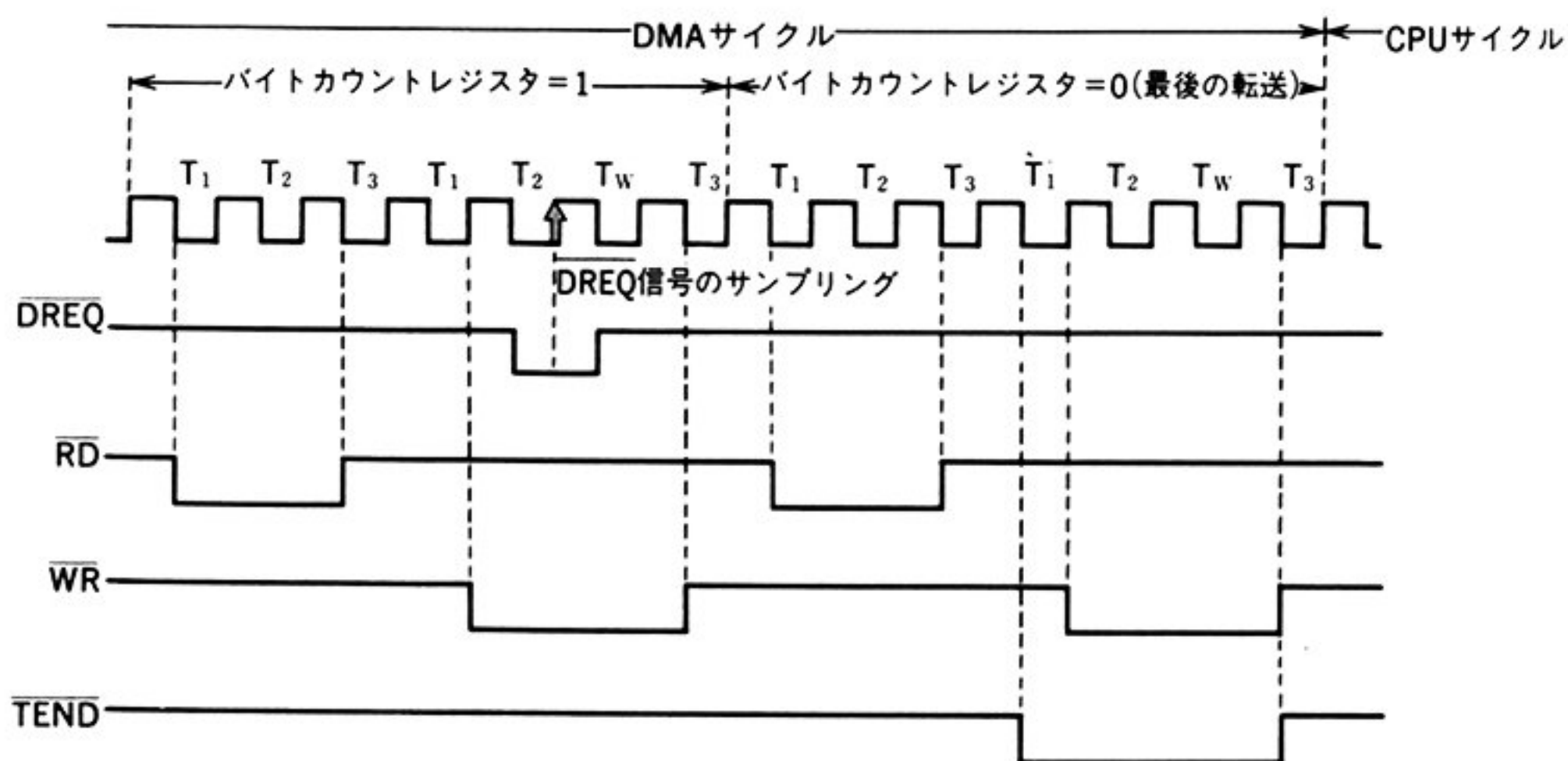



図 11・7 $\overline{\text{TEND}}$信号の出力

タイミングにアクティブになっている必要がありますが，エッジセンスの場合立下りエッジがラッチされ，サンプリングのタイミングまで保持されています．

$\overline{\text{DREQ}}$信号は，CPUの通常のマシンサイクル中は，マシンサイクルごとのT_3サイクルの1つ前のクロックの立上りでサンプリングされ，DMAサイクル中は，DMAライトサイクルのT_3サイクルの1つ前のクロックの立上りでサンプリングされます．$\overline{\text{DREQ}}$信号のサンプリングタイミングを**図11・6**に示します．

最大転送レートは，メモリサイクル3サイクル，I/Oサイクル4サイクル（1WAIT挿入）の計7サイクルで1バイトのデータ転送が終了するため，1.1Mバイト/s(f=8MHz)を得ることができます．

転送が終了すると$\overline{\text{TEND}}$信号が出力され，I/Oに対しDMA転送が終了したことを知らせます．$\overline{\text{TEND}}$信号は，最後の1バイトの転送のライトサイクル中アクティブ（Low状態）になります．$\overline{\text{TEND}}$信号の出力タイミングを**図11・7**に示します．

11・4 チャネルの優先順位

DMA転送の開始要求が，チャネル0，1とも同時に発生した場合，チャネル0が優先的に受け付けられます．一方，どちらかのチャネルが連続してDMA転送を行っている場合，後から転送要求の発生したチャネルは動作することができません．これには次の2つの場合が考えられます．

(1) チャネル0がメモリ↔メモリで動作中の場合（バーストモード，サイクルスチールモードのいずれの場合も含む）…図11・8の場合

(2) チャネル0または1が，メモリ↔I/Oを連続して行っている場合…図11・9の場合

いずれの場合も動作中のチャネルが優先され，後から転送要求が発生したチャネルは動作することができません．

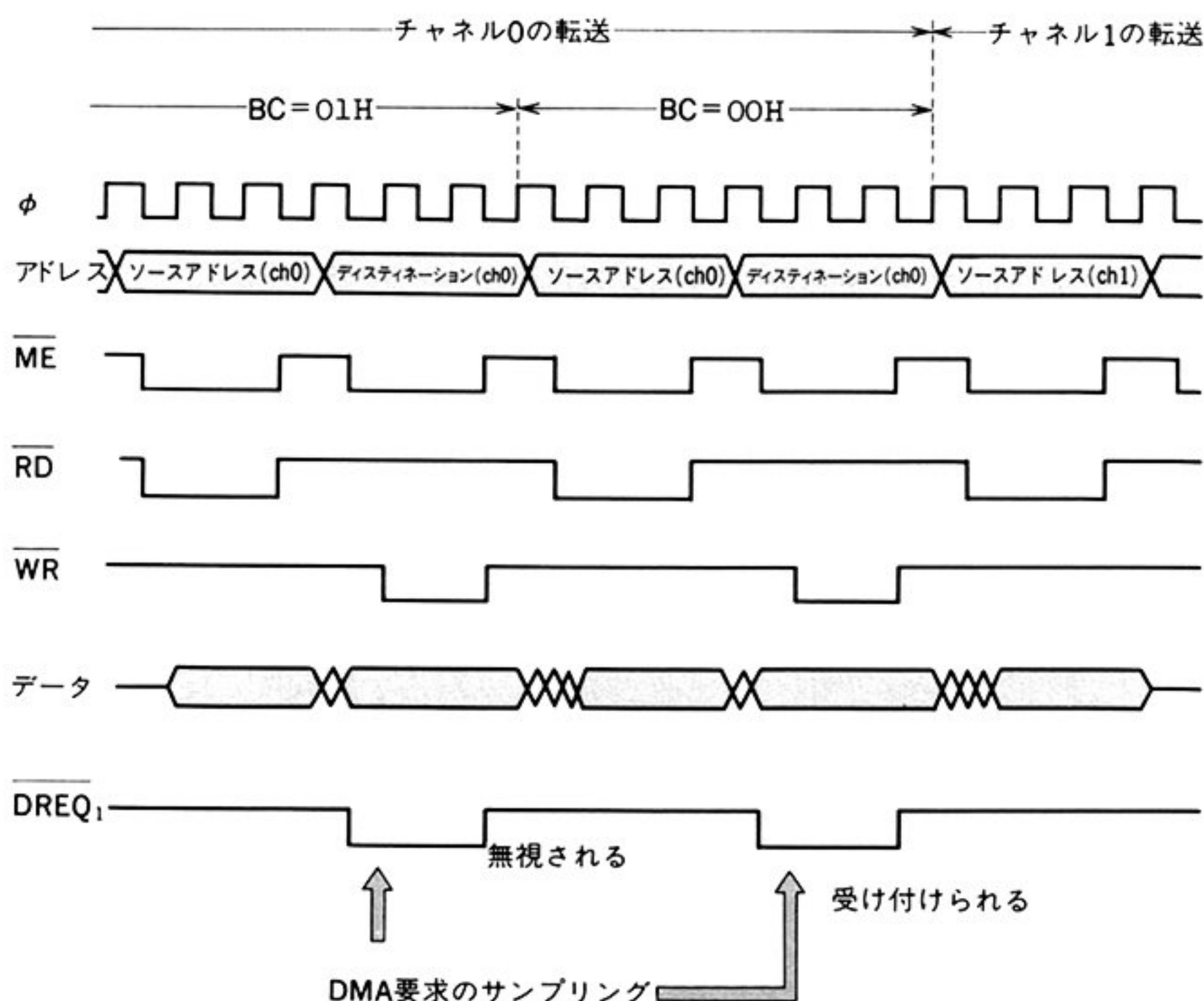



図11・8 チャネル0（メモリ↔メモリ）とチャネル1の競合

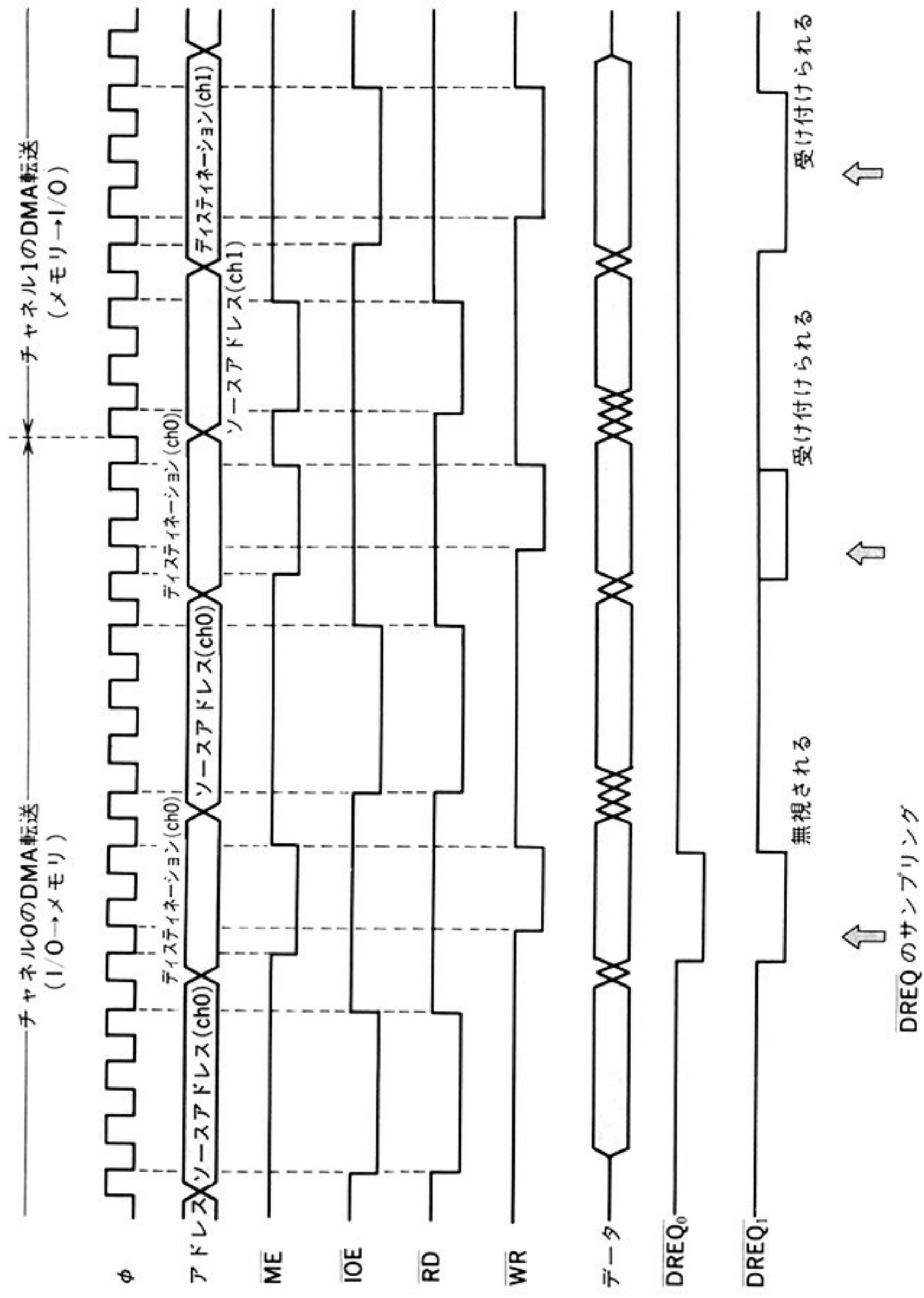



図 11・9 チャネル0(メモリ↔I/O)とチャネル1の競合

ただし，メモリ↔I/O の転送の場合，動作中のチャネルの転送要求がとだえ，この間にもう一方のチャネルの転送要求が発生すれば，後から要求の発生したチャネルが動作可能となります．また，DMA の転送要求信号（$\overline{\mathrm{DREQ}}$ 信号）はエッジセンスとレベルセンスを選択可能ですが，エッジセンスを選択した場合，この要求は実際に転送が行われるまで 1 回だけ保持されます．したがって，動作中のチャネルの転送終了後，以前に入力され無視されていたエッジの $\overline{\mathrm{DREQ}}$ 入力がサンプリングされ，DMA 転送を行います．これに対し，レベルセンスでは要求を保持しませんから，動作中のチャネルの転送終了後，$\overline{\mathrm{DREQ}}$ 端子がアクティブ（Low 状態）になっていないと DMA 転送は行われません．

なお，メモリ↔メモリの転送の場合，転送が開始されると，サイクルスチールモードの場合もバーストモードの場合でも，チャネル 0 の転送が終了するまでチャネル 1 は動作することができません．

11・5　DMAC とブロック転送命令

64180 の DMA 転送では，メモリ↔メモリのデータ転送が可能です．一方，64180 はデータ転送を行うため，ブロック転送命令をもっています．ここでは，DMA 転送とブロック転送命令の転送速度の比較をしてみましょう．

DMAC のアドレスレジスタの設定には，メモリアドレスと I/O アドレスが 1 バイト設定ごとにプラス 1 される，OTIMR 命令を使用します．ただし，設定値はメモリ空間上のテーブルに連続して配置されているとします．次に，OUT0 命令を用いて，DMA ステータスレジスタ，DMA モードレジスタ，DMA/WAIT コントロールレジスタの設定を行います．このプログラムを実行するには，201 ステート必要です．したがって，物理空間上でのメモリ↔メモリの n バイトの転送を行うと，6n+201 ステートとなります．

次に，ブロック転送命令を使用した場合を考えます．HL レジスタに転送元のメモリアドレス，DE レジスタに転送先のメモリアドレス， BC レジスタに転

```
LD  HL, CNST0       9 ←────────── アドレスレジスタに設定する設定値が格納さ
LD  BC  CNST1       9               れているメモリアドレスの先頭番地
OTIMR             {16 不成立 16×7+14  B：設定するレジスタのバイト数(8 バイト)
                  {14 成立            C：DMAC のレジスタの先頭アドレス(20H 番地)
LD  A, CNST2        6
OUT0(DCNTL), A     13
LD  A, CNST3        6
OUT0(DMODE), A     13
LD  A, CNST4        6
OUT0(DSTAT), A     13                DMA イネーブル
──────────────────────────────────────────────────────────
                  201 ステート
```

（1） DMA 転送

```
LD  HL, CNST5       9               転送元のメモリアドレス
LD  DE, CNST6       9               転送先のメモリアドレス
LD  BC, CNST7       9               転送バイト数
LDIR              {14 不成立 14(n−1)+12
                  {12 成立
──────────────────────────────────────────────────────────
                  14n+25 ステート
```

（2） ブロック転送

図 11・10　DMAC とブロック転送命令

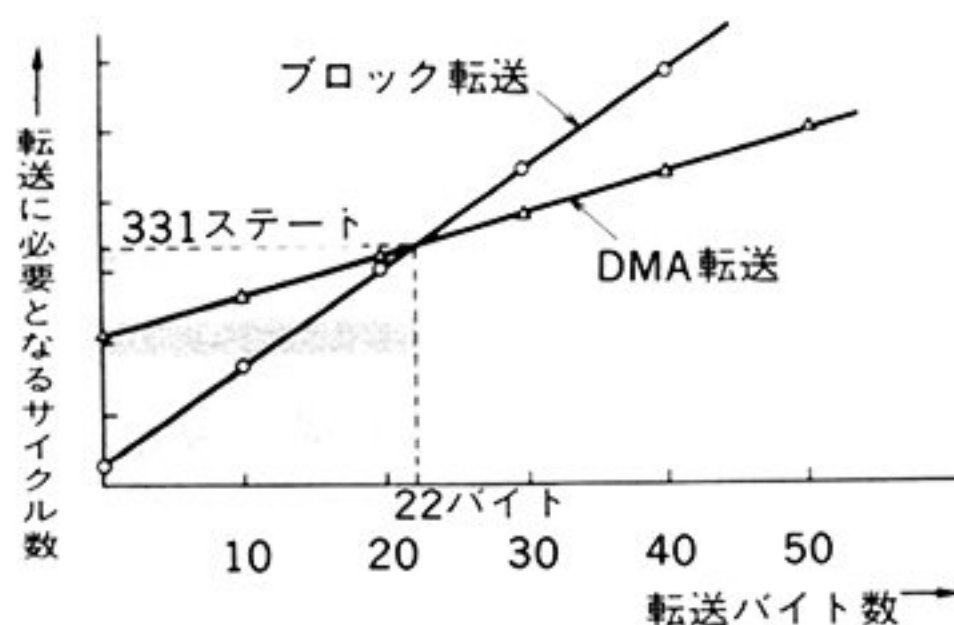



図 11・11　DMA 転送とブロック転送のサイクル数

表 11・3　DMAC とブロック転送命令の比較

	DMAC	ブロック転送命令
転送の対象となる空間	物理空間（1 M バイト）	論理空間（64 K バイト）
連続転送可能なバイト数	64 K バイト	←
1 バイトの転送に要するステート数	6 ステート	14 ステート
n バイトの転送に必要なステート数	6n＋201 ステート	14n＋25 ステート

送バイト数を設定し，ブロック転送命令（LDIR 命令など）を実行します．n バイトの転送を実行するには，14n＋25 ステート必要となります．実際のプログラムを**図 11・10** に示します．

したがって，**図 11・11** に示すとおり 22 バイト以上の転送では，DMAC を使用したほうが速く転送が行えます．

表 11・3 に DMAC とブロック転送命令の比較を示します．DMA 転送とブロック転送命令の大きな違いは，前者は物理空間上のデータ転送を対象としているのに対し，後者は論理空間上でのデータ転送を対象としている点です．たとえば，論理空間上で連続しているコモンエリア 1 とバンクエリアが，物理空間上では不連続な場合もあります．このような場合は，DMAC よりもブロック転送命令を使用するほうが得策です．これに対し，論理空間上にない物理空間上のデータを参照する場合は，DMAC により直接データを論理空間上にストアするのが便利です．

11・6 DMACの使用例

〔1〕 **メモリ↔メモリ**　DMACのチャネル0でメモリ↔メモリの転送を行うためには，次のような手続きを行います．

(1) 転送元の転送開始アドレスを SAR0（I/O 空間の 20，21，22H）に設定

(2) 転送先の転送開始アドレスを DAR0（I/O 空間の 23，24，25H）に設定

(3) 転送するデータのバイト数を BCR0（I/O 空間の 26，27H）に設定

(4) メモリ↔メモリの指定，アドレスの増減，およびサイクルスチール，バーストモードの転送モードを DMA モードレジスタ（I/O 空間の 31H）に設定

(5) DMA ステータスレジスタ（I/O 空間の 30H）に設定を行い，DMA転送をイネーブルにする．また DMA 転送終了後，割込みを発生することも可能

DMA 転送をイネーブルにしたI/O ライトサイクル終了後，次の命令の1マシンサイクルを実行し，DMA 転送が開始されます．DMAC はレジスタが多く（ただ

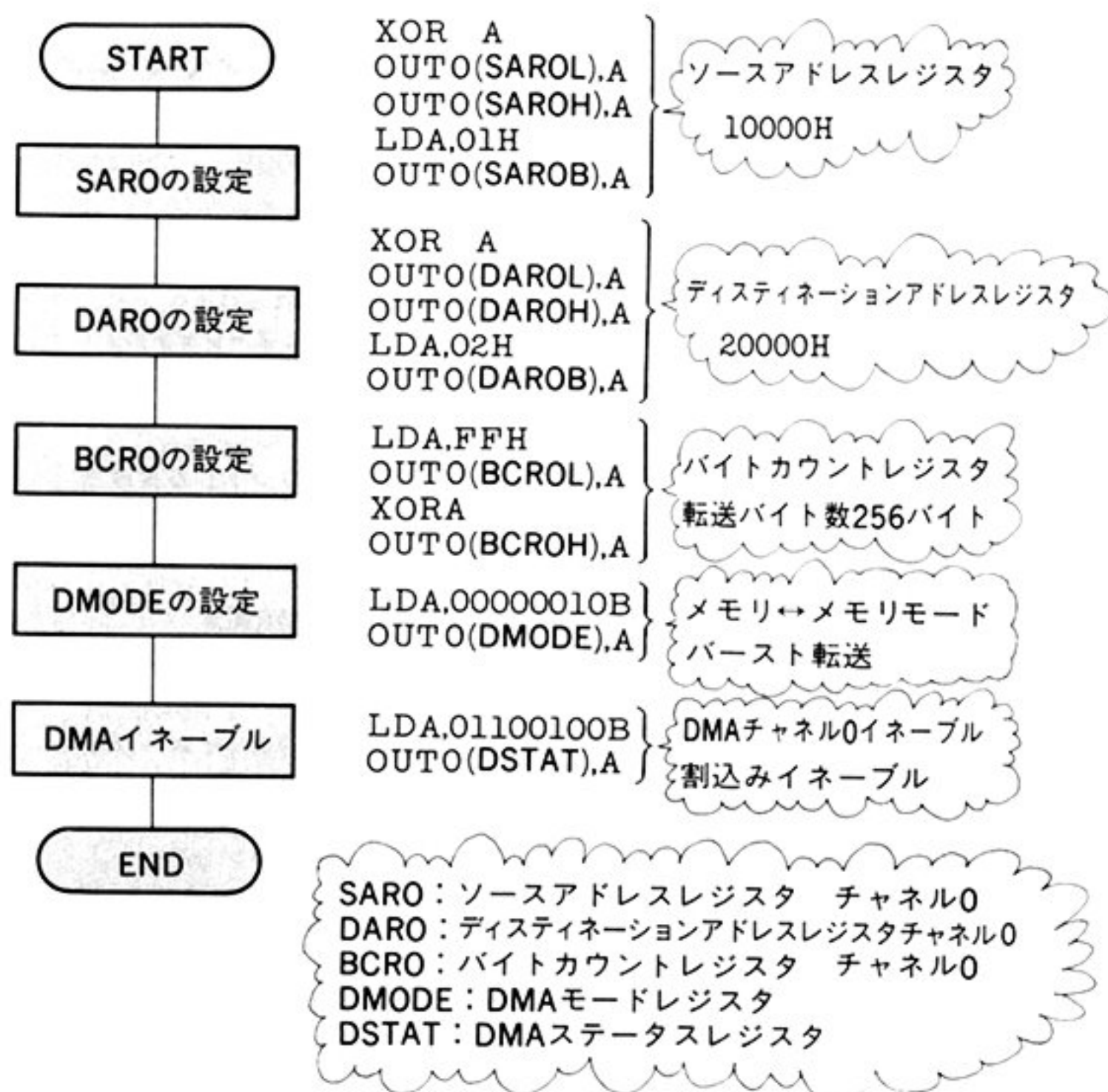



図 11・12 メモリ↔メモリ転送のプログラム例（チャネル0）

し連続に配置されている)，64180の新規命令のOTIMR命令やOUTO命令を使用すると便利です．**図11・12**にフローチャートと実際のプログラムを示します．

〔2〕 **メモリ↔I/O**　チャネル0，1ともメモリ↔I/Oの転送が可能です．ここでは，チャネル1でメモリ→I/Oの転送を行う手順を示します．

(1) 転送元のメモリアドレスをMAR1 (I/O空間の28，29，2AH) に設定

(2) 転送先のI/OアドレスをIAR1 (I/O空間の2B，2CH) に設定

(3) 転送するデータのバイト数をBCR1 (I/O空間の2D，2EH) に設定

(4) $\overline{DREQ}$信号のセンス (エッジセンス，レベルセンス) の指定，ソース，ディスティネーションの指定，およびメモリアドレスの増減の指定を，DMA/WAITコントロールレジスタ (I/O空間の32H) に設定

(5) DMAステータスレジスタに設定を行い，DMAをイネーブルにする．また割込みも使用可能

この状態で$\overline{DREQ}$信号がアクティブ (Low状態) になると，メモリ→I/OのDMA転送が行われます．**図11・13**にフローチャートとプログラム例を示します．

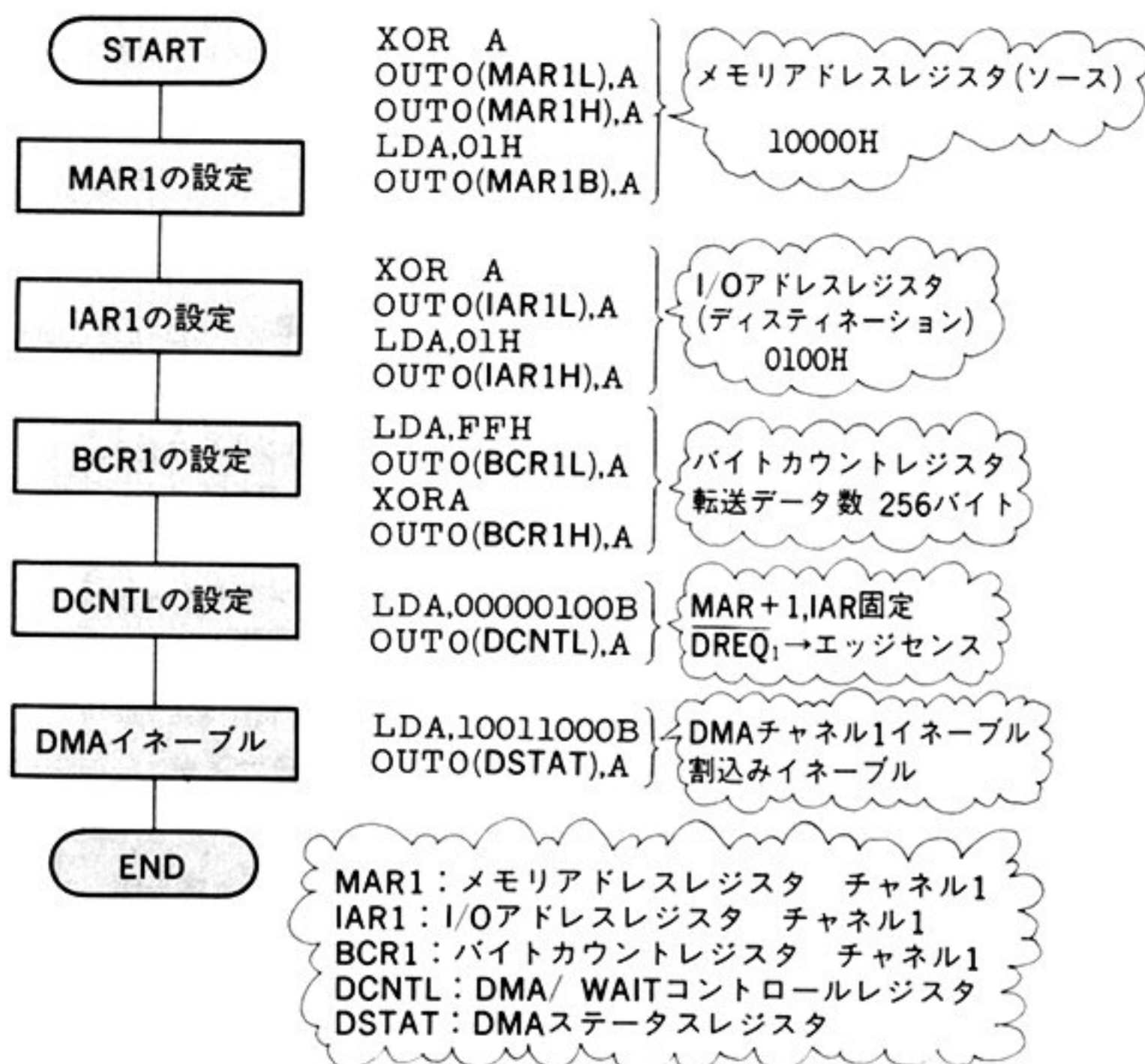



図 11・13　メモリ↔I/O転送のプログラム例 (チャネル1)

第III編

応　　用

第III編では，64180を使って実際にシステムを組む場合のインタフェース例を述べます．64180は周辺LSIの機能を取り込んでいますが，マルチチップマイコンのため，メモリ，I/Oを接続してシステムを組みます．そこで，12章ではSRAM, DRAMなどのメモリとのインタフェース方法，13章では各種周辺LSIとのインタフェース方法を述べます．14章では，64180の新規命令の使用例を述べます．15章では64180を使用した具体的なシステムとして，180カードパソコンについて述べます．180カードパソコンは葉書サイズのボードに，64180, 256 KDRAM, EPROM, FDCを搭載したCP/M PLUSシステムです．

12. メモリインタフェース

64180はMMUを通して，1Mバイトもの大容量メモリを管理することができます．この章では，64180とメモリ（SRAM, DRAM, EPROM）との具体的なインタフェース例を述べます．64180はリフレッシュコントローラを内蔵しているため，DRAMとも容易に接続できます．また，最近注目されてきているDRAMのように大容量メモリでありながら，SRAMのように使いやすい擬似SRAMとのインタフェース例も述べます．

12・1 SRAMの接続

64180はSRAMと簡単に接続できます．HM62256（32Kワード×8 bit）との接続の場合，$\overline{CS}$信号はアドレスと$\overline{ME}$信号により生成します．$\overline{OE}$信号は$\overline{RD}$信号に，$\overline{WE}$信号は$\overline{WR}$信号に接続します．図12・1では，SRAMは64180の物理アドレス08000H番地〜0FFFFH番地に配置します．

2^{15} 32,768　2^{16}=65535

メモリアクセス時のクリティカルパスは，オペコードフェッチ時です．オペコードフェッチ時はT_3サイクルの立上りでデータが取り込まれます．ϕ=6.144 MHz時，ウェイトサイクルなしでアクセスタイムの許容範囲は144 ns（$\overline{CS}$信号デコード回路の遅延を含む）以下で，アクセスタイム120 nsのSRAMが使えます．ϕ=8 MHz時では，ウェイトサイクルなしでアクセスタイムの許容範囲は107 ns（$\overline{CS}$信号デコード回路の遅延を含む）以下で，アクセスタイム85 nsのSRAMが使えます．

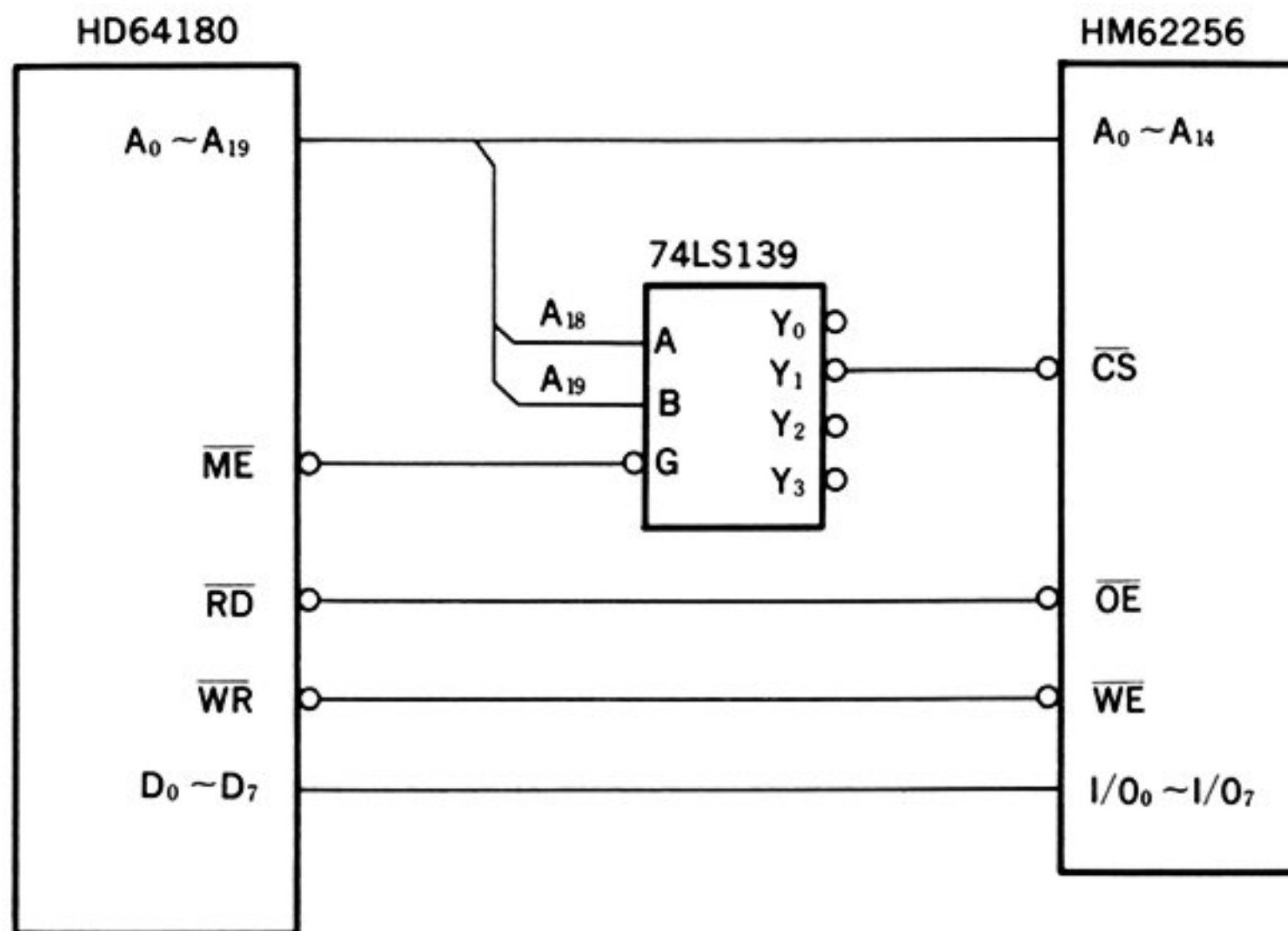



図 12・1　SRAMの接続

74LS139

G	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

12・2 擬似SRAMの接続

擬似SRAMは，SRAMと同様に使えること，かつDRAMのように安価であることからポピュラーになりつつあります．64180に擬似SRAMを接続した例を図12・2に示します．擬似SRAMは，DRAMと同様にリフレッシュを行わなければなりませんが，64180はリフレッシュ機能を内蔵しているため，回路が簡単になります．HM658128は9bitのリフレッシュアドレスが必要ですが，64180は8bitリフレッシュアドレスしか出力しないため，本来なら1bit分追加する外部回路が必要です．その対策として，擬似SRAMがもっているリフレッシュアドレスを必要としない**オートリフレッシュモード**を使用します．このモードではHM658128内部のリフレッシュカウンタにより，$\overline{CE}$＝'High'時に$\overline{RFSH}$＝'Low'にするだけで，内部でリフレッシュが実行されます．したがって64180を使う場合，8ms内に512回リフレッシュサイクルが入るようにリフレッシュサイクルの間隔を設定するだけで，擬似SRAMのリフレッシュが行えます．たとえばϕ＝6.144 MHzの場合，リフレッシュ間隔を80ステートに設定します．

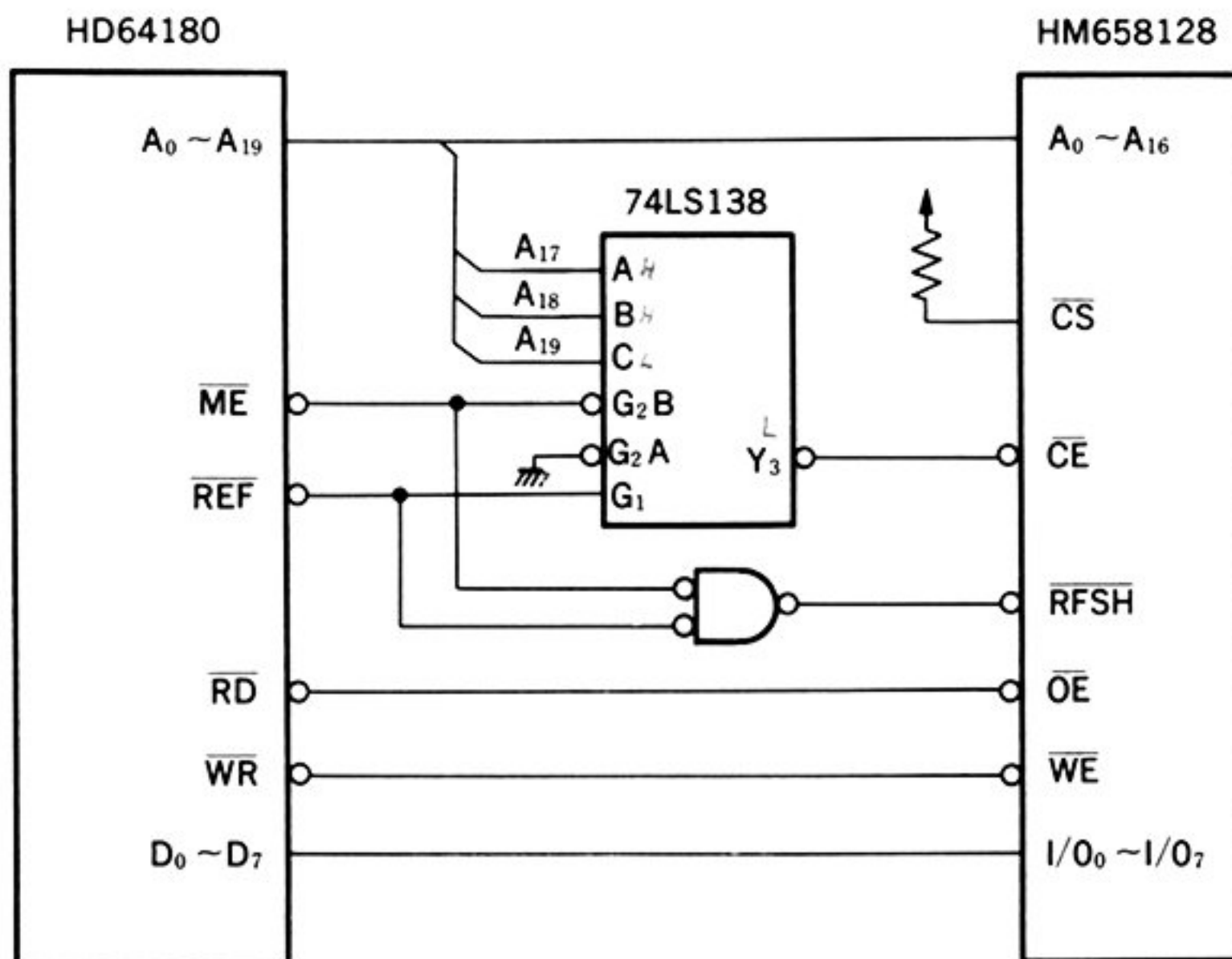



図 12・2 擬似SRAMの接続

12・3 DRAMの接続

DRAMを接続するためには，アドレスのマルチプレクサ，$\overline{RAS}$，$\overline{CAS}$信号の生成，リフレッシュサイクルの発生が必要です．64180はリフレッシュコントローラを内蔵しているため，通常のマイコンシステムで必要なリフレッシュコントロール用回路（リフレッシュアドレス発生用カウンタ，CPUへのバス権要求回路など）が不要になります．64180に256KDRAMを接続した例を**図12･3**に示します．

DRAMのアドレスは，ロウアドレスとカラムアドレスを時分割で入力します．そのため，アドレスマルチプレクサを使います．$\overline{RAS}$信号の立下りでロウアドレスを確定しますが，64180の$\overline{ME}$信号をそのまま$\overline{RAS}$信号として使用できます．この場合，$\overline{ME}$信号の‘High’時間およびアドレスセットアップ時間に注意します．**図12･4**にDRAMアクセスタイミングを示します．アドレスマルチプレクサ切替え信号と$\overline{CAS}$信号は，通常ディレイ回路で複雑になるのですが，64180

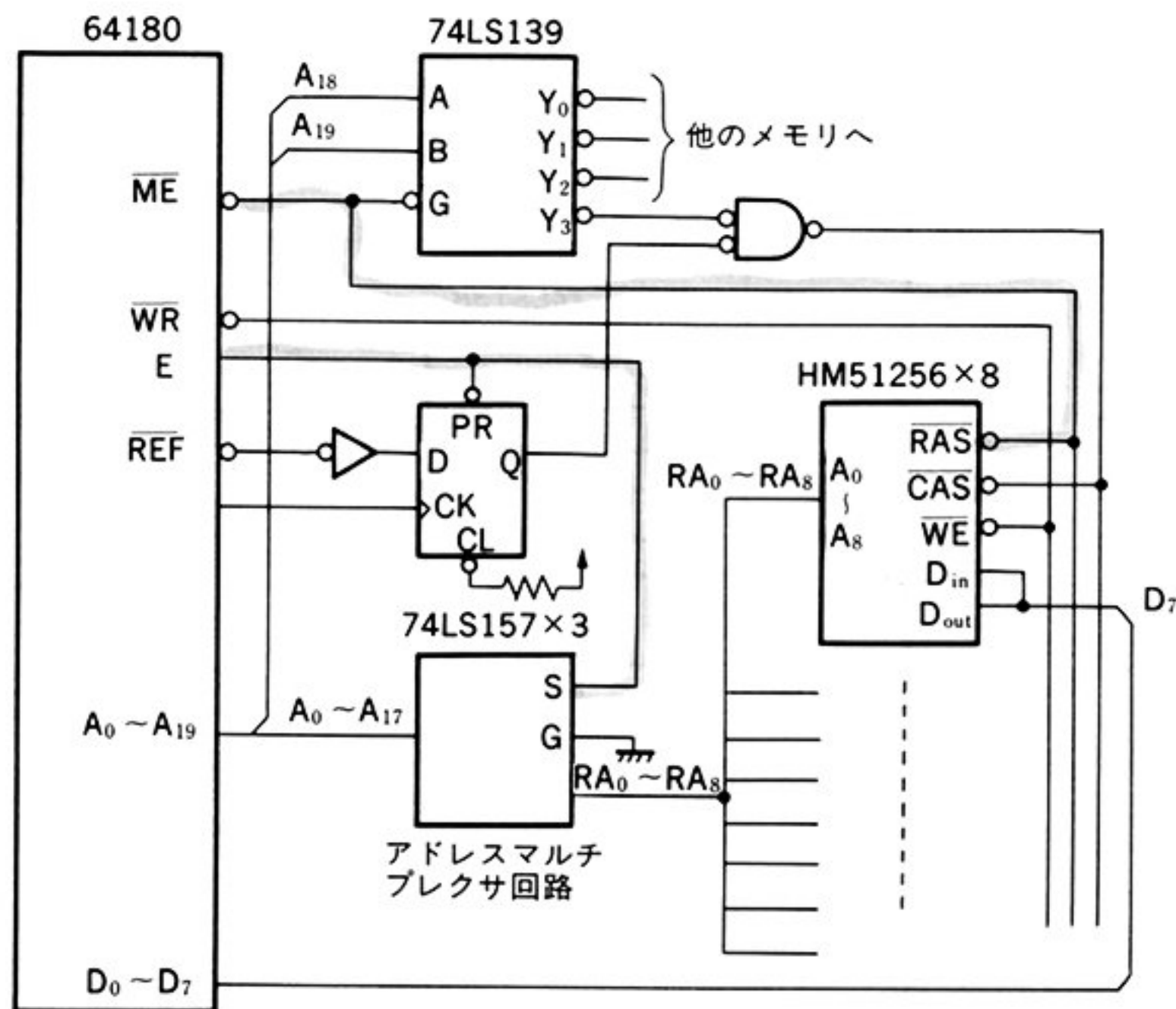



図 12・3 DRAMの接続例

から出力されているE信号を使用すると簡単な回路で生成できます．本来E信号は，68 系周辺 LSI を接続するための信号ですが，DRAMコントロールに流用しています．$\overline{\text{ME}}$ 信号による $\overline{\text{RAS}}$ 信号の立下りでロウアドレスを確定し，E 信号の立上りでカラムアドレスに切り替え，次の ϕ 信号の立上りで $\overline{\text{CAS}}$ 信号を立ち下げ，カラムアドレスを確定します．この例では $\overline{\text{CAS}}$ 信号が遅れるため，1ウェイトサイクル挿入します．

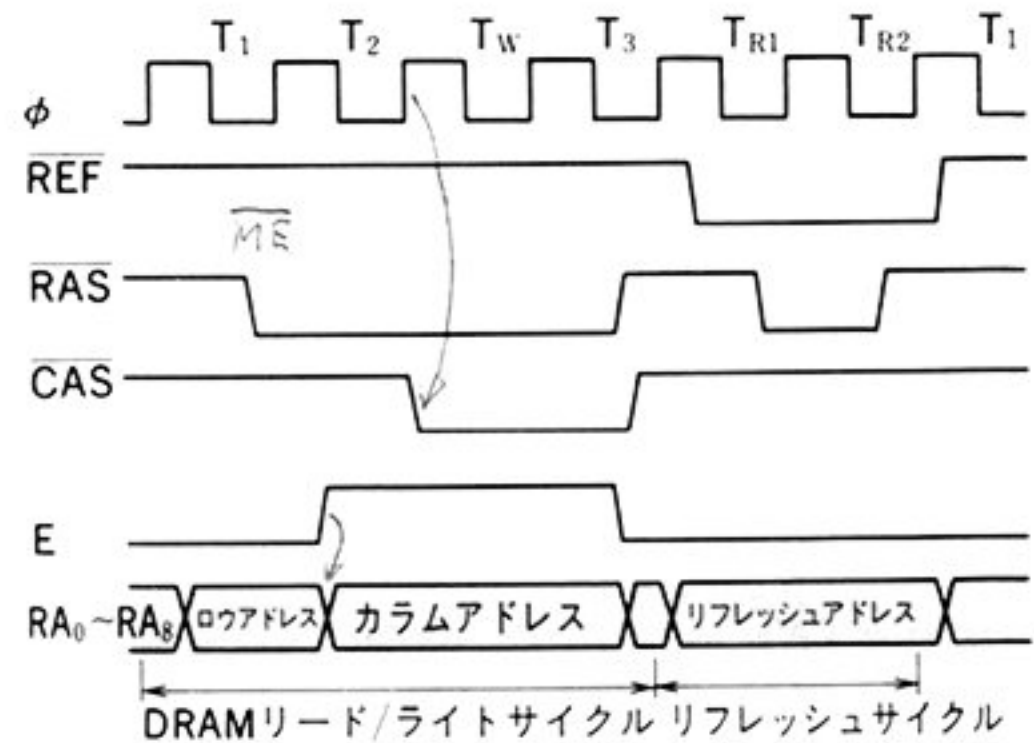



図 12・4 DRAM アクセスタイミング

DRAMのリフレッシュは，$\overline{\text{RAS}}$ オンリリフレッシュモードで行います．この場合リフレッシュアドレスを入力しますが，64180 内蔵のリフレッシュコントローラが，8 bit のリフレッシュアドレスを発生します．リフレッシュサイクルの間隔は，動作周波数によりリフレッシュコントロールレジスタを設定して決めます．たとえば，ϕ=6.144 MHz の場合，80 ステート間隔に設定します．64180 内蔵のリフレッシュコントローラは 8 bit のリフレッシュアドレスを出力するため，256 リフレッシュサイクル/4 ms の 256KDRAM しか接続できません．512 リフレッシュサイクル/8 ms の 1MDRAM を接続するためには，**図 12・5** に示すようなリフレッシュアドレスを 1 bit 追加する回路を外付します．

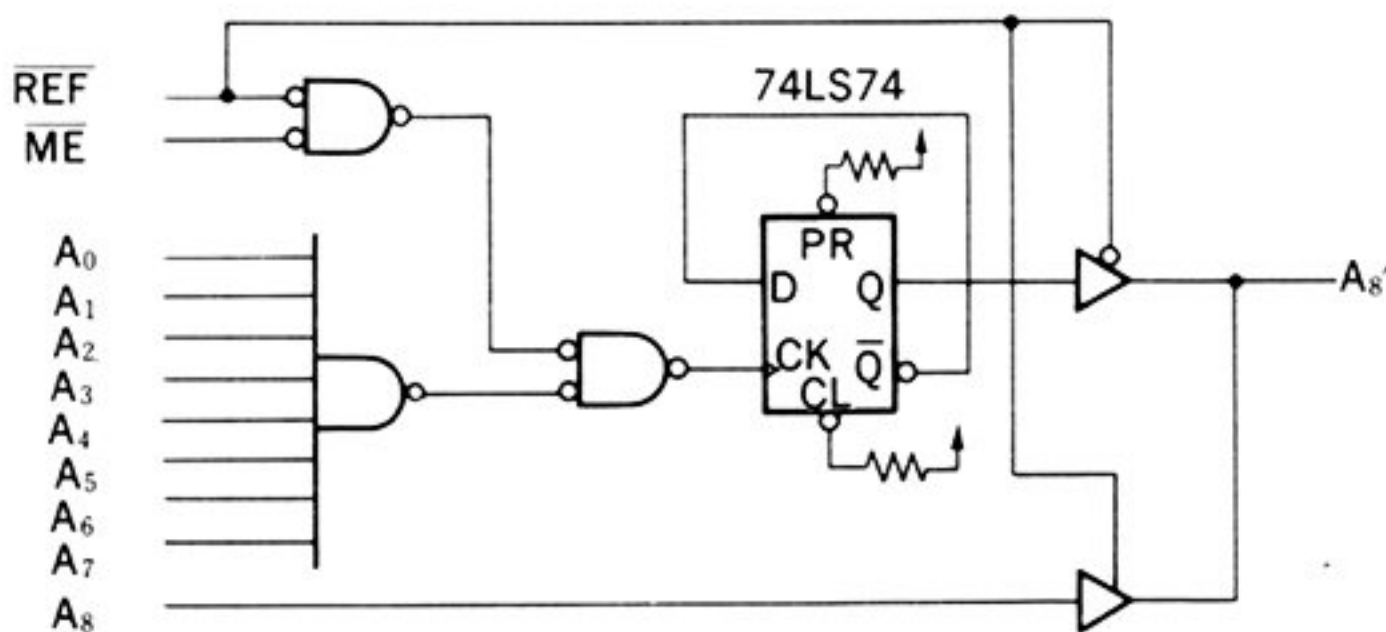



図 12・5 1M DRAM 接続時のリフレッシュアドレス追加回路

12・4 EPROMの接続

EPROMの$\overline{CE}$信号はアドレスと$\overline{ME}$信号で生成します．EPROMはリードオンリのため，誤ってEPROMにデータをライトしようとしても，データが衝突しないように64180の$\overline{RD}$信号をEPROMの$\overline{OE}$信号に接続してください．

SRAMの場合と同様に，オペコードフェッチ時のアクセスタイムの許容範囲はϕ=6.144 MHzで144 ns以下，ϕ=8 MHzで107 ns以下（いずれもウェイトサイクルなしでデコード回路の遅延を含む）になります．最近では，アクセスタイムが100 ns以下の高速EPROMが出始めていますが，大部分のEPROMのアクセスタイムは200 ns以上です．そのため，ウェイトサイクルを挿入しなければなりません．64180内蔵のウェイトコントロール機能を使うと，外付ハードウェアなしでEPROMをアクセスできます．

図12・6では，消費電流を抑えるため，アクセスタイミングを$\overline{CE}$，$\overline{OE}$両信号で制御しています．EPROMはリードオンリのため，$\overline{CE}$=‘Low’固定とし，$\overline{OE}$信号のみで制御することができます．この場合，アクセスタイムを短縮でき，ウェイトサイクルを挿入しなくてもよくなります．ただし，$\overline{CE}$=‘Low’のままだと消費電流は増えたままです．特にCMOSタイプのEPROMでは，顕著な差が現れます．

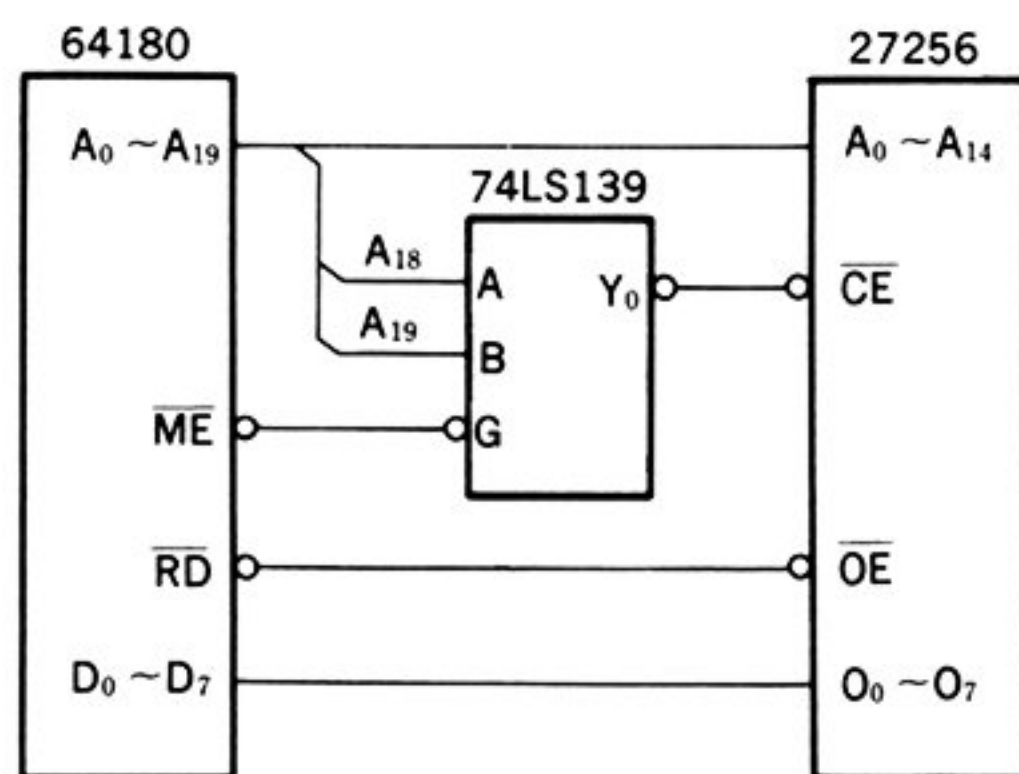



図 12・6　EPROMの接続

12・5 $\overline{\text{WAIT}}$信号発生回路

64180内蔵のウェイトコントロール機能は，外付ハードウェアなしで中低速メモリをアクセスできるので便利です．ただし，メモリアクセス時に必ず設定した数だけウェイトサイクルが入るため，EPROMとSRAMが混在した場合など，EPROMをアクセスしたときはウェイトサイクルが入り，SRAMをアクセスしたときはウェイトサイクルが入らないようにすることは，ソフトウェアだけでは不可能です．64180の$\overline{\text{WAIT}}$信号にLowを入力すると，その期間ウェイトサイクルを挿入できます．**図12•7**のような，EPROMの$\overline{\text{CE}}$信号が‘Low’になったときだけ$\overline{\text{WAIT}}$信号を‘Low’にする外付回路を付加すると，選択的なウェイトサイクル挿入ができます．**図12•8**にウェイトステートタイミングを示します．

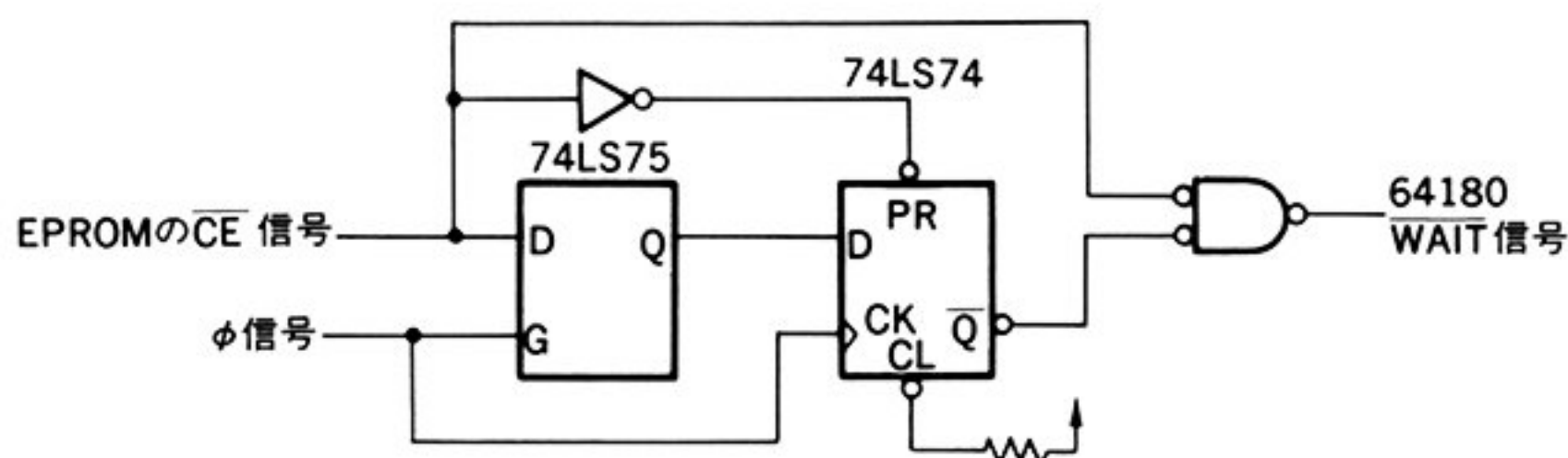



図 12・7 $\overline{\text{WAIT}}$信号発生回路

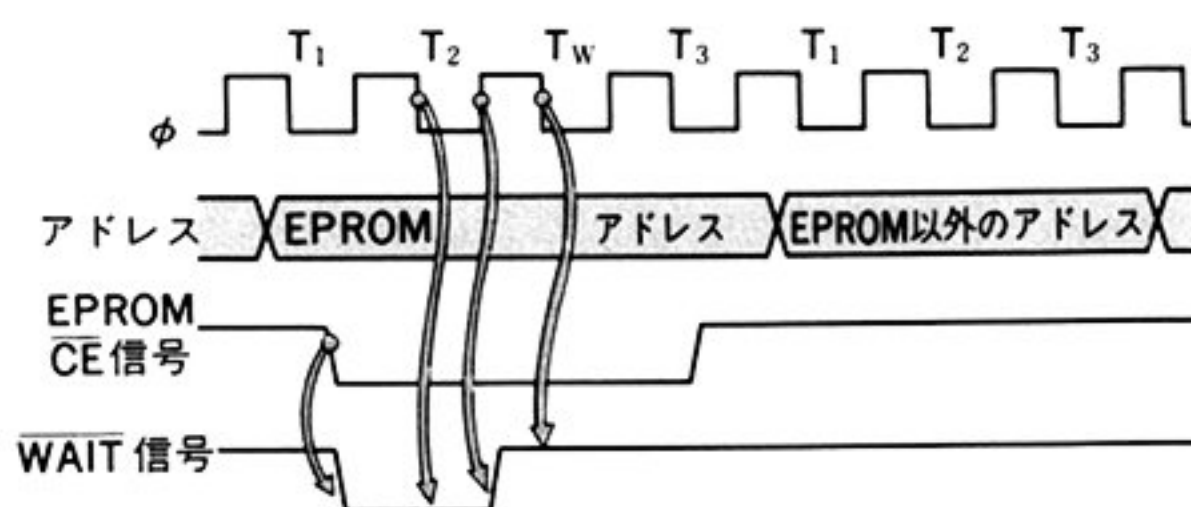



図 12・8 ウェイトステートタイミング

13. I/O インタフェース

64180は，DMAC，ASCI，タイマなどの周辺機能を内蔵していますが，その他に各種I/Oを外付できます．8255，8251などの80系周辺LSI，Z80-PIO，SIOなどのZ80系周辺LSI，6845，6321などの68/63系周辺LSIとのインタフェース例を述べます．Z80系周辺LSIを接続する場合，64180Zバージョンを使用します．68/63系周辺LSI用には，64180から出力されているE信号を使用します．

13・1 80系周辺LSIとの接続

80系周辺LSIは，$\overline{CS}$，$\overline{RD}$，$\overline{WR}$信号で制御します．図13・1では，80系タイマ8253を例にとり説明します．$\overline{RD}$，$\overline{WR}$信号に対する$\overline{CS}$信号のセットアップ時間を確保するため，$\overline{CS}$信号はアドレスのみから生成します．80系周辺LSI用の$\overline{RD}$，$\overline{WR}$信号は，64180の$\overline{IOE}$信号と$\overline{RD}$，$\overline{WR}$信号のANDをとって生成します．

80系周辺LSIでは$\overline{CS}$，$\overline{RD}$，$\overline{WR}$信号の幅でアクセスが規定されています．そのため，64180の動作周波数が高くても，80系周辺LSIをアクセスするときだけウェイトサイクルを挿入することにより64180を接続できます．64180は内蔵ウェイトコントローラにより，ソフトウェアでI/O空間アクセス時に1～4ウェイトサイクル挿入できます．この機能を使用すると，64180が高速動作時も外部ウェイト回路なしで80系周辺LSIを接続できます．ϕ=6.144 MHzで8253が5 MHzバージョンの場合，1ウェイトサイクル挿入します．図13・2にタイミ

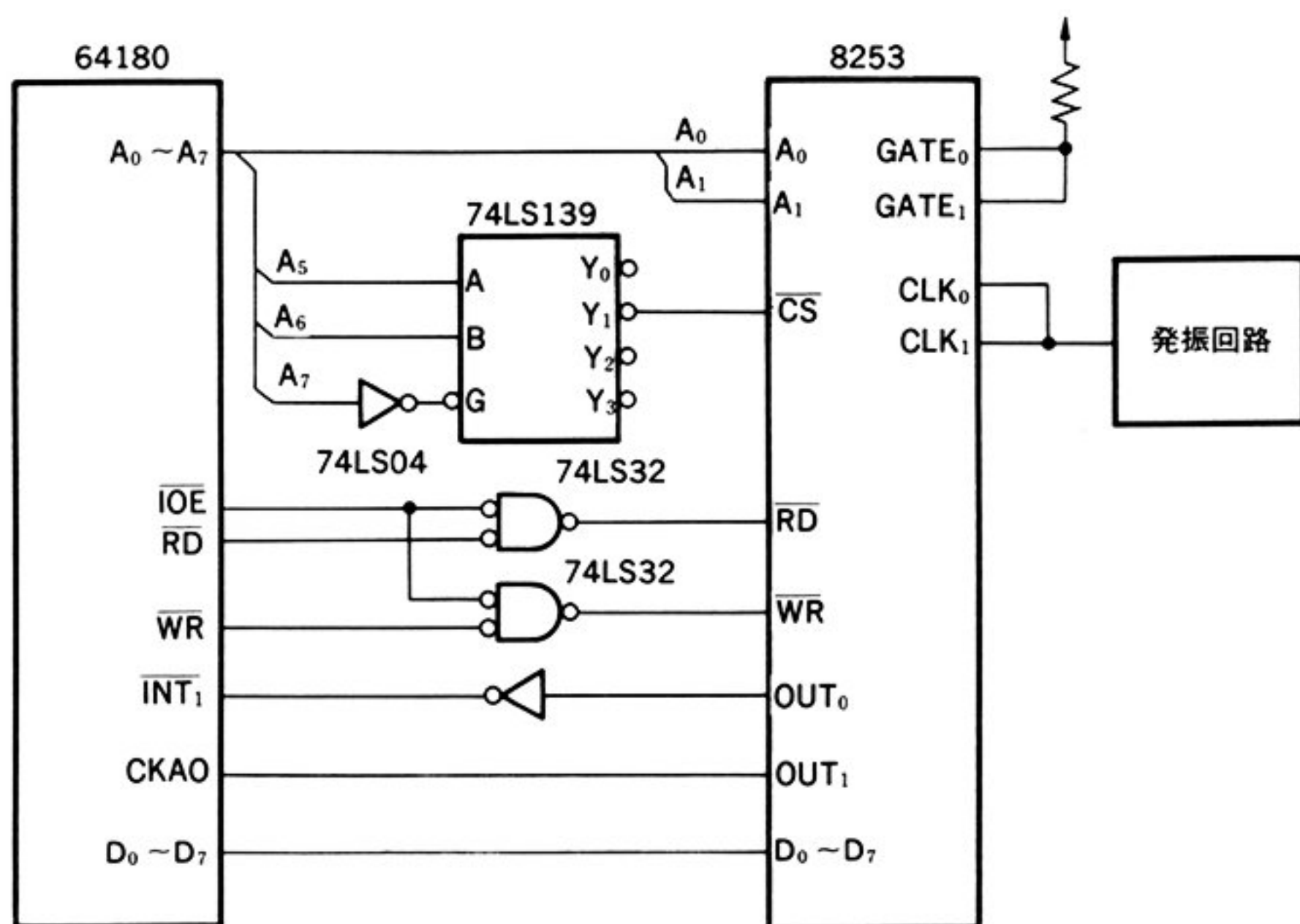



図13・1 80系周辺LSIの接続例

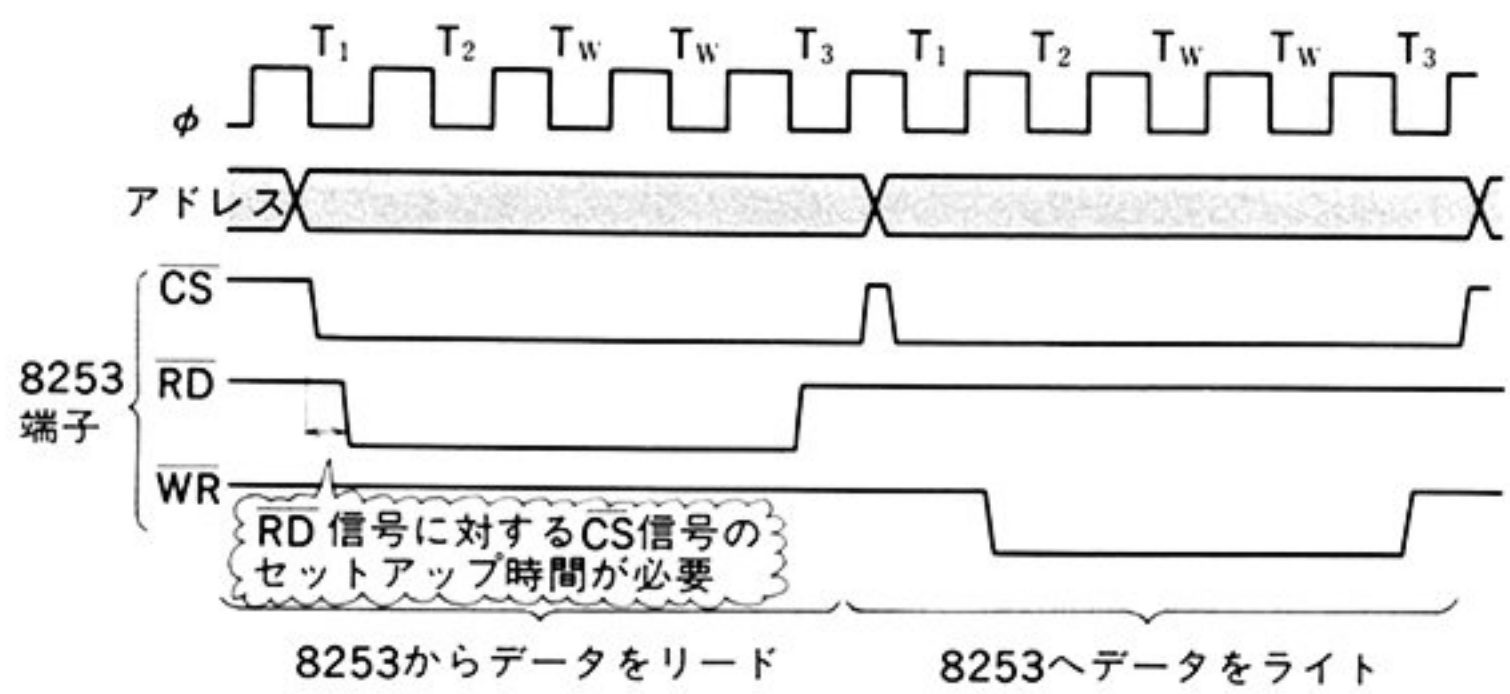



図 13・2　8253のアクセスタイミング

ング図を示します．なお，この場合，各信号のセットアップ，ホールド時間に注意しなければなりません．

周辺LSIの種類

8ビットマイコンの周辺LSIは，80系，Z80系，68/63系に分けられます．**表13・1**に周辺LSIの種類を示します．通常，バスタイミングの違いから，80系・Z80系と68系の周辺LSIを混同して使用することは困難でした．しかし，64180は68系周辺LSI用にE信号を出力しているため，68系周辺LSIを容易に接続できます．

表 13・1　周辺LSIの種類

	80系	Z80系	68/63系
パラレル I/O	8255	Z80-PIO	6821/6321
シリアル I/O	8251	Z80-SIO	6850/6350
タイマ	8253	Z80-CTC	6840/6340
その他	8237 (DMAC)	Z80-DMA (DMAC)	6845 (CRTC)

13・2 Z80周辺LSIの接続

64180はZ80上位コンパチブルになっています．ただし，Z80周辺LSIと**デイジーチェイン接続**する場合，**64180Z**を用います．64180R1を使用する場合ハードウェアに制限があり，外付回路が必要です．64180Zでもリセット直後は64180R1と同じ信号タイミングです．そのため，動作モードコントロールレジスタを設定して，Z80用の信号タイミングに変更します．**表13・2**に動作モードコントロールレジスタの設定方法を示します．なお，64180とZ80の信号名が少し違います．64180の$\overline{IOE}$，$\overline{LIR}$信号がZ80の$\overline{IORQ}$，$\overline{M1}$信号に相当します．

表13・2 64180Zでの動作モードコントロールレジスタの設定

No.	Z80周辺LSI			OMCRへのビットセット		
	デイジーチェイン	CTC	PIO	LIRE	$\overline{LIRTE}$	$\overline{IOC}$
1	有	有	有	0	"0"をライト*	0
			無	0	—	0
		無	有	0	"0"をライト*	0/1
			無	0	—	0/1
2	無	有	—	1	—	0
		無	—	1	—	0/1

* Z80PIOのレジスタ設定後1回実行

Z80周辺LSIを使用する場合，割込みの優先順位を決定できるデイジーチェイン接続に特徴があります．デイジーチェインは，Z80周辺LSIの$\overline{IEI}$，$\overline{IEO}$端子を接続して実現します．**図13・3**ではSIOが一番優先順位が高くなっています．デイジーチェイン接続を行うときに注意しなければならないのは，64180ZのASCI，DMACなどの内蔵周辺機能とはデイジーチェイン接続できないことです．そのため，デイジーチェイン接続の特徴になっている優先順位の高いデバイスが割込みルーチンを実行中は，優先順位の低いデバイスは割込みを要求できないという機能が64180Z内蔵の周辺機能に対して使えません．また，デイジーチェイン接続されているZ80周辺LSIからの割込みルーチンを実行中に，64180Z内蔵の周辺機能から割込み要求があり，その割込みルーチンの最後でRETI命令を実行すると，Z80周辺LSIは自分の割込みルーチンが終了したと勘違いし

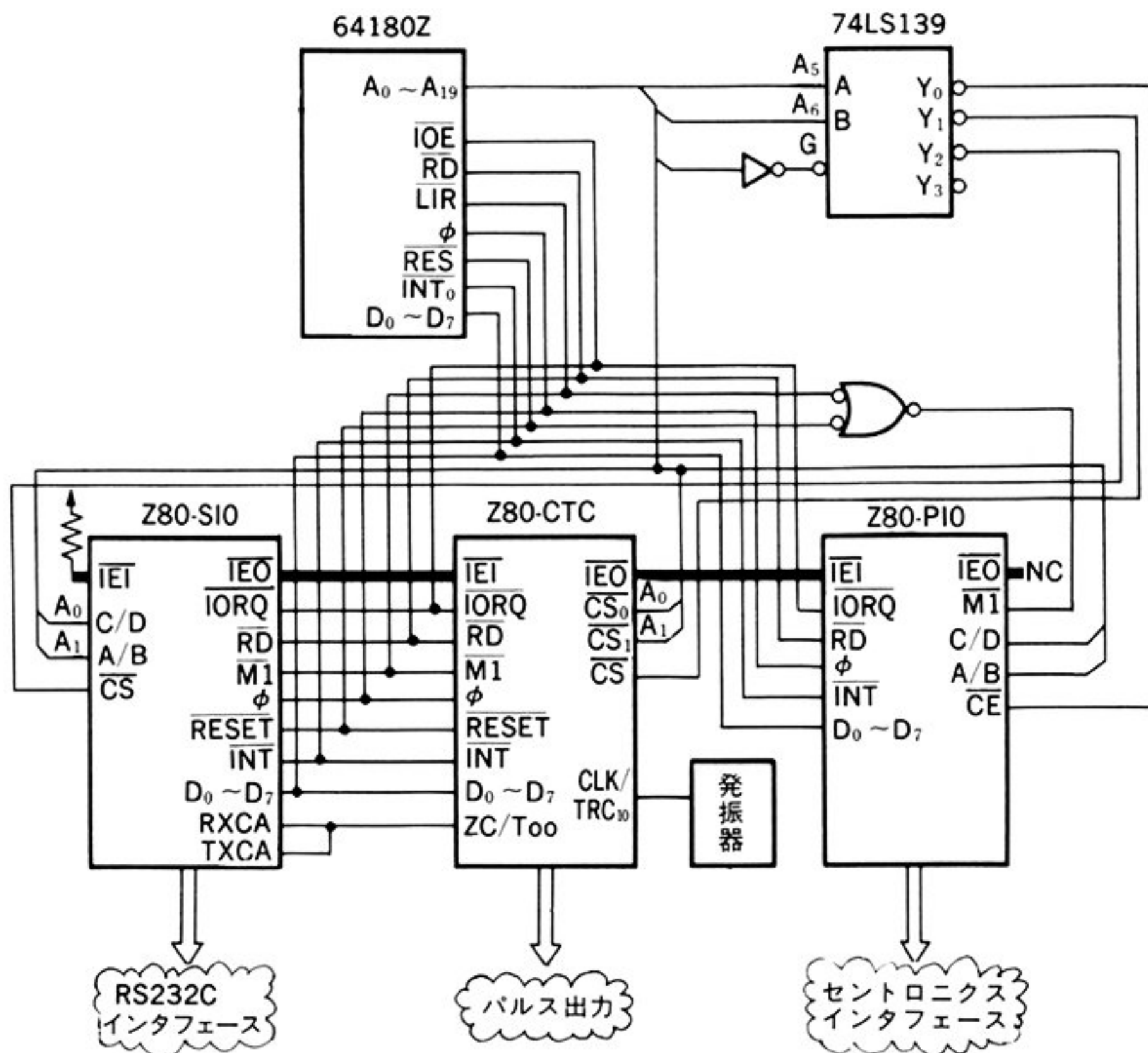



図 13・3　Z80周辺LSI接続例

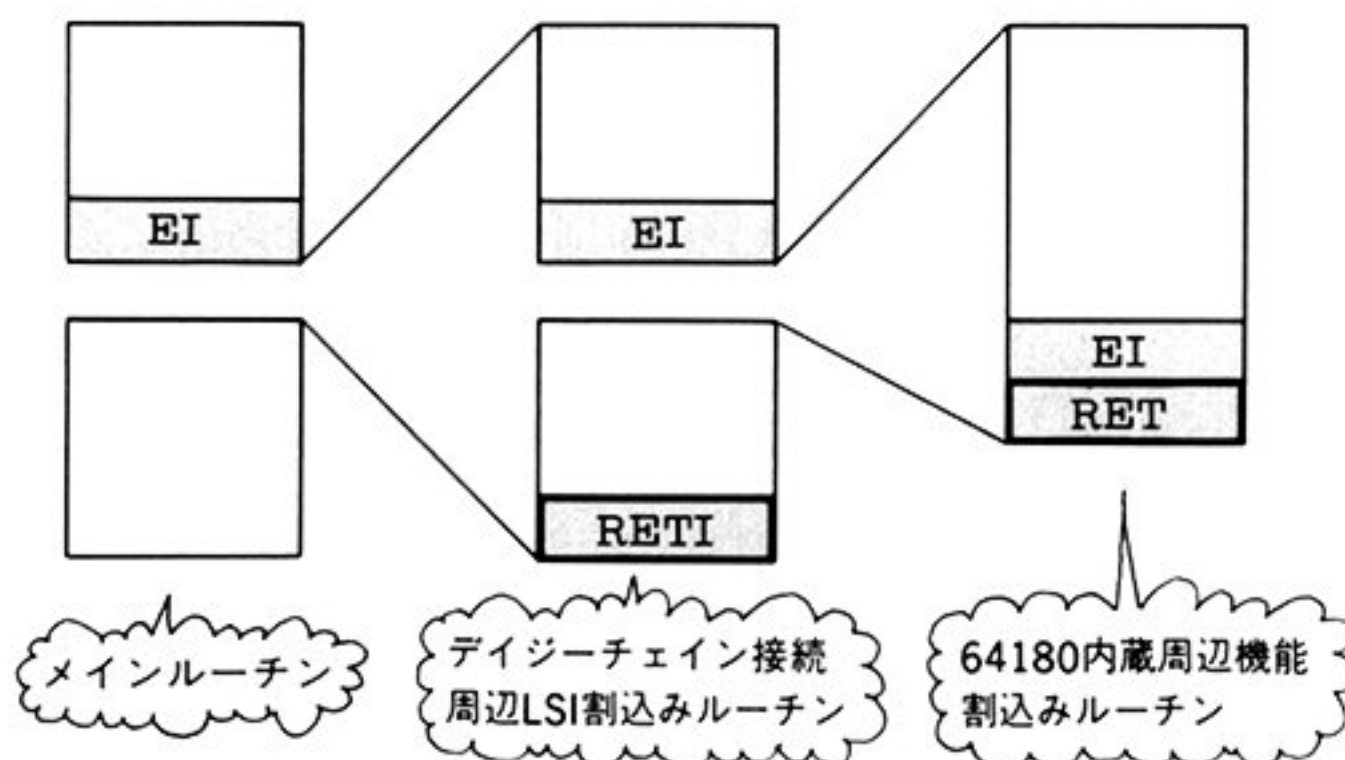



図 13・4　デイジーチェイン接続時の割込みルーチンからの復帰

て，$\overline{\text{IEO}}$ 信号を 'High' にし優先順位の低い周辺 LSI への割込み禁止を解除します．この勘違いは，デイジーチェイン接続されていない割込みルーチンで RETI 命令を実行したのが原因です．そこで，デイジーチェイン接続を行っている場合，デイジーチェイン接続をしていない 64180Z 内蔵の周辺機能や他の周辺 LSI の割込みから復帰するときは，**図13・4** のように RET 命令を使用します．

通常，64180 ではデイジーチェイン接続を行わないほうが，システム構築が考えやすくなります．

13・3　68系周辺LSIとの接続

64180は68系周辺LSI用に**E信号**を出力しています．E信号を使えば，68系周辺LSIを80系CPUの64180にダイレクトに接続できます．**図13・5**にHD63B21を接続した例を示します．68系周辺LSIを接続する場合に注意しなければならないのは，E信号の‘High’出力期間です．64180のバスは非同期バスであるため，メモリ，I/Oのリード・ライト間隔が一定ではありません．そのため，E信号の波形が不規則になります．**表13・3**に，64180の各状態におけるE信号の出力タイミングを示します．E信号の‘High’期間が最少幅になるのは，I/Oライトサイクル時です．たとえば，HD63B21のE信号‘High’幅は最低220 ns以上必要です．そのため，64180の動作周波数はϕ=4.5 MHz以下で使用するか，ウェイトサイクルを挿入します．

また，E信号の出力形がduty 50％でないため，周辺LSIがE信号を基にして外部とインタフェースしている場合，注意が必要です．たとえば，HD63B21では，ハンドシェイク制御出力から64180のEクロックサイクル幅がそのまま出

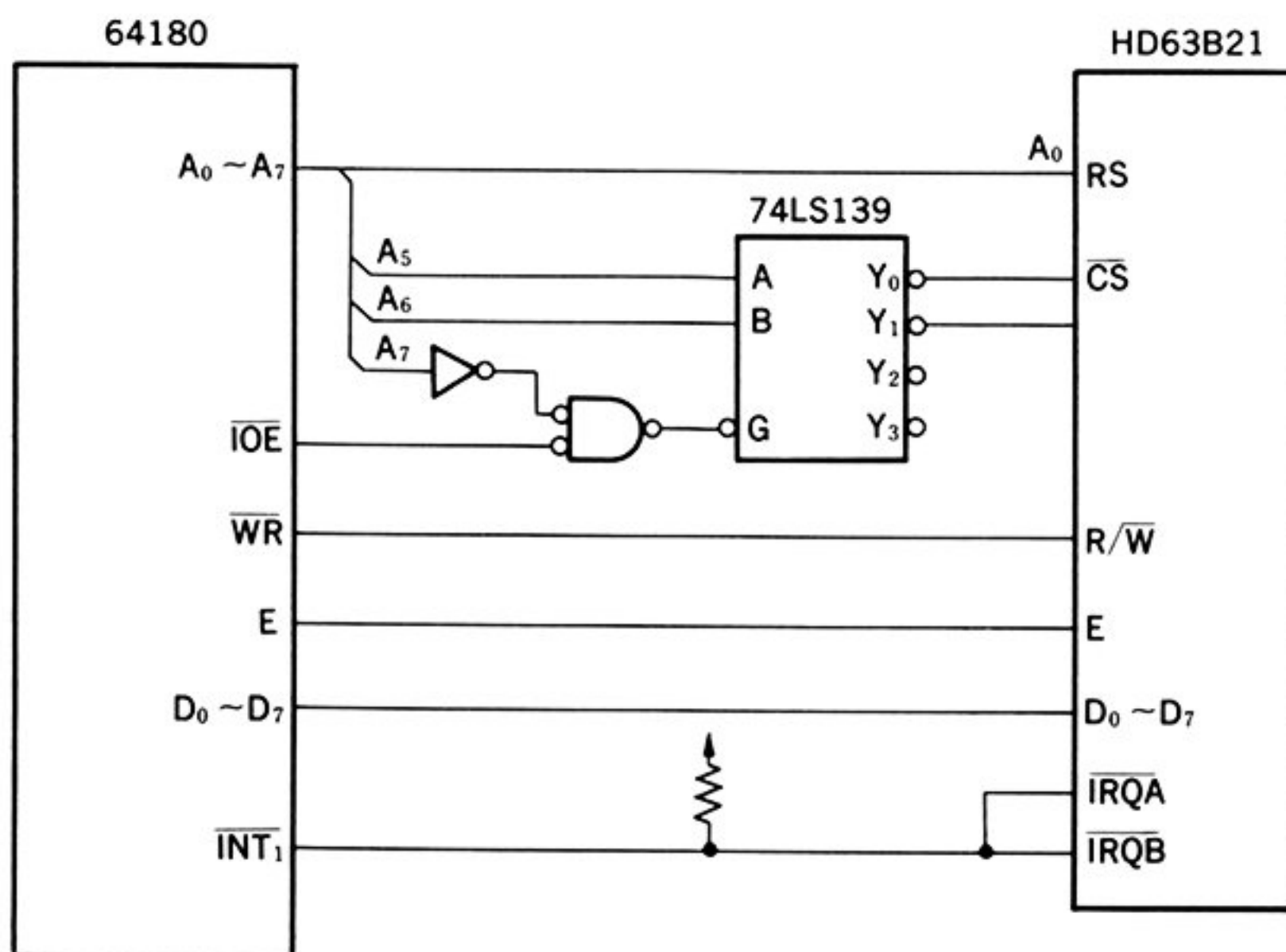



図 13・5　68系周辺LSIの接続例

表 13・3 各状態におけるE信号の出力タイミング

バス状態	E信号の'High'タイミング	E信号の'High'期間
オペコードフェッチ メモリリード/ライトサイクル	T_2↗～T_3↘	1.5φ+nφ
I/O リードサイクル	I'st T_W↗～T_3↘	0.5φ+nφ
I/O ライトサイクル	I'st T_W↗～T_3↗	n・φ
$\overline{NMI}$アクノレッジ第1マシンサイクル	T_2↗～T_3↘	1.5φ
$\overline{INT_0}$, $\overline{INT_1}$, $\overline{INT_2}$ 内部割込み アクノレッジ第1マシンサイクル	I'st T_W↗～T_3↘	0.5φ+nφ
Bus Release, Sleep System Stopモード	φ↘～φ↘	2φ または 1φ

n：WAITステートの挿入数

力されます．ところが，同じ68系周辺LSIである6845は内部レジスタのアクセスのみにE信号を使用しているため，E信号の'High'期間のみ注意すればいいことになります．

割込みを使う場合は，HD63B21の割込み出力信号はオープンドレイン出力となっているため，プルアップ抵抗を接続し，レベルセンス，ベクタ方式の外部割込み信号の$\overline{INT_1}$，または$\overline{INT_2}$に接続すればいいでしょう．

13・4 FDC の接続

64180 に FDC（Floppy Disk Controller）を接続する場合，64180 と FDC 間のデータ転送方法として，**プログラム転送を使う方法**と **DMA 転送を使う方法**があります．通常，他の CPU ではプログラム転送を使うため，他のプログラムを実行することができません．また，DMA 転送を使うと高価な DMAC が必要になります．それに対して，64180 は DMAC を内蔵しているため，外付 DMAC なしで DMA 転送を行うことができ，プログラム実行のスループットを上げることができます．**図 13・6** に，64180 と FDC の HD63265 の接続例を示します．

HD63265 は高精度データセパレータ回路を内蔵しているため，外付 VFO 回路が不要であり，標準の FDC とソフトウェアコンパチであり，また，80 系，68 系両バスインタフェースを備えた FDC です．HD63265 の DMA 制御信号の $\overline{\text{DREQ}}$，$\overline{\text{TEND}}$ 信号は，64180 の $\overline{\text{DREQ}_1}$，$\overline{\text{TEND}_1}$ 信号に接続します．なお，$\overline{\text{DACK}}$ 信号に対応する信号を 64180 はもっていません．なぜなら，64180 内蔵の DMAC は，メモリアドレスと I/O アドレスを別々に出力するデュアルアドレス方式の DMA 転送をサポートしているからです．そのため，シングルアドレス方式の DMA 転送

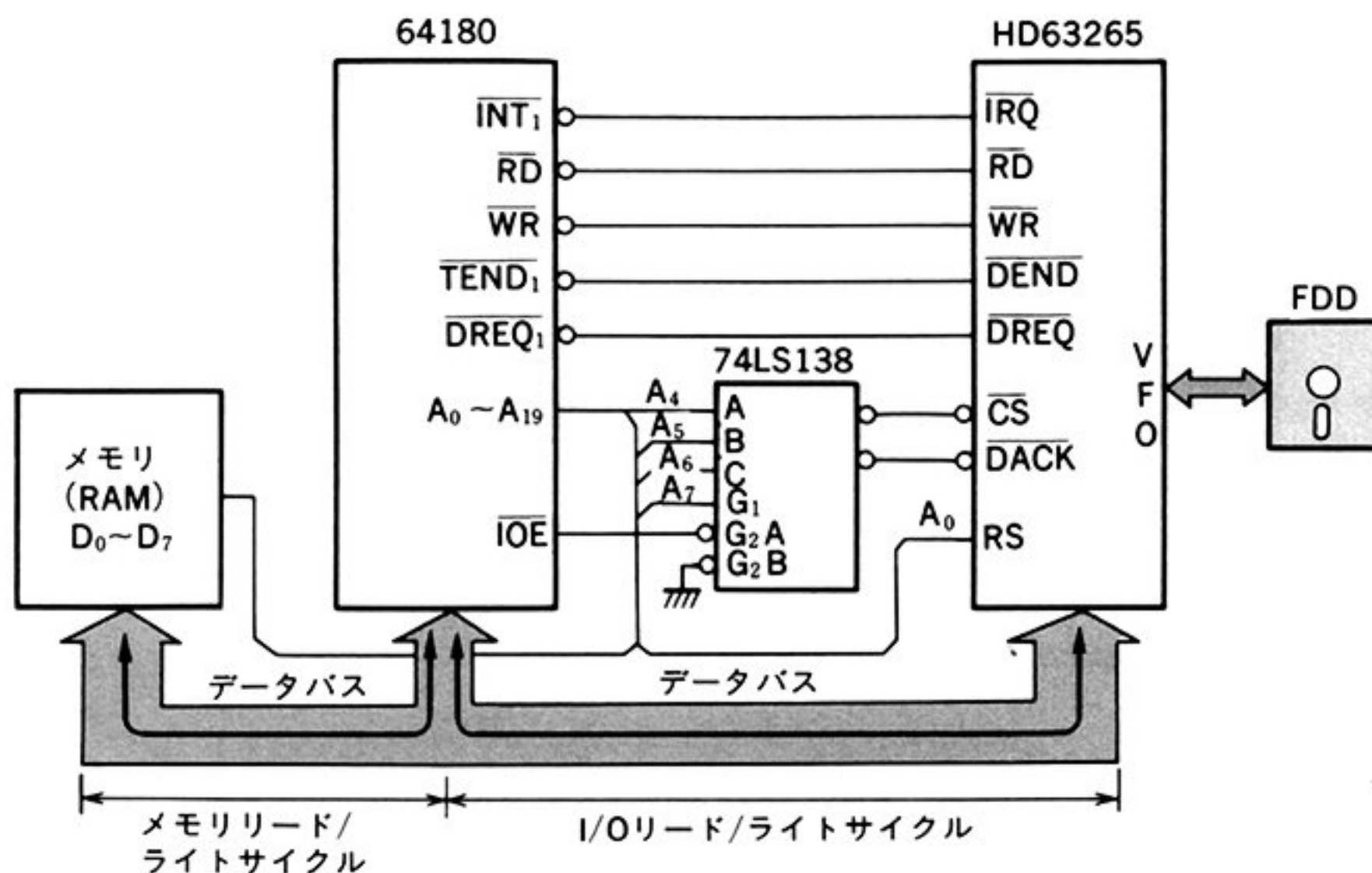



図 13・6 FDC の接続例

表 13・4　FDCアドレスマップ

I/Oアドレス	内　　容
80H	FDC（HD63265） ステータスレジスタ
81H	FDC（HD63265） データレジスタ
90H	FDC（HD63265） $\overline{\mathrm{DACK}}$信号用（DMA時）

用のFDCと接続する場合，$\overline{\mathrm{DACK}}$信号を別の方法で生成します．64180はI/Oのリード/ライト時と同様に，DMA転送時もI/Oアドレスを出力します．そこで，DMA転送用のI/Oアドレスを設定しておき，$\overline{\mathrm{CS}}$信号と同様にデコード信号を$\overline{\mathrm{DACK}}$信号に入力します(**表13・4**)．**図13・7**に，64180とHD63265のDMA転送タイミングを示します．

$\overline{\mathrm{DREQ_1}}$信号はレベル入力とエッジ入力が選択可能ですが，HD63265とのインタフェースでは**エッジ入力**を選択します．DMAライトサイクル時$\overline{\mathrm{DACK}}$信号が'Low'になると，HD63265は$\overline{\mathrm{DREQ}}$信号を'High'にします．64180はT_3ステートの前の，ステートのϕの立上りで$\overline{\mathrm{DREQ_1}}$信号をサンプリングしているため，レベル入力を選択すると$\overline{\mathrm{DREQ_1}}$信号が'High'になる前にサンプリングしてしまい，DMA転送が余分に行われます．エッジ入力を選択しておくと，立下りエッジを検出しないとDMA転送を行わないため，$\overline{\mathrm{DREQ}}$信号の立上りが遅れてもだいじょうぶです．

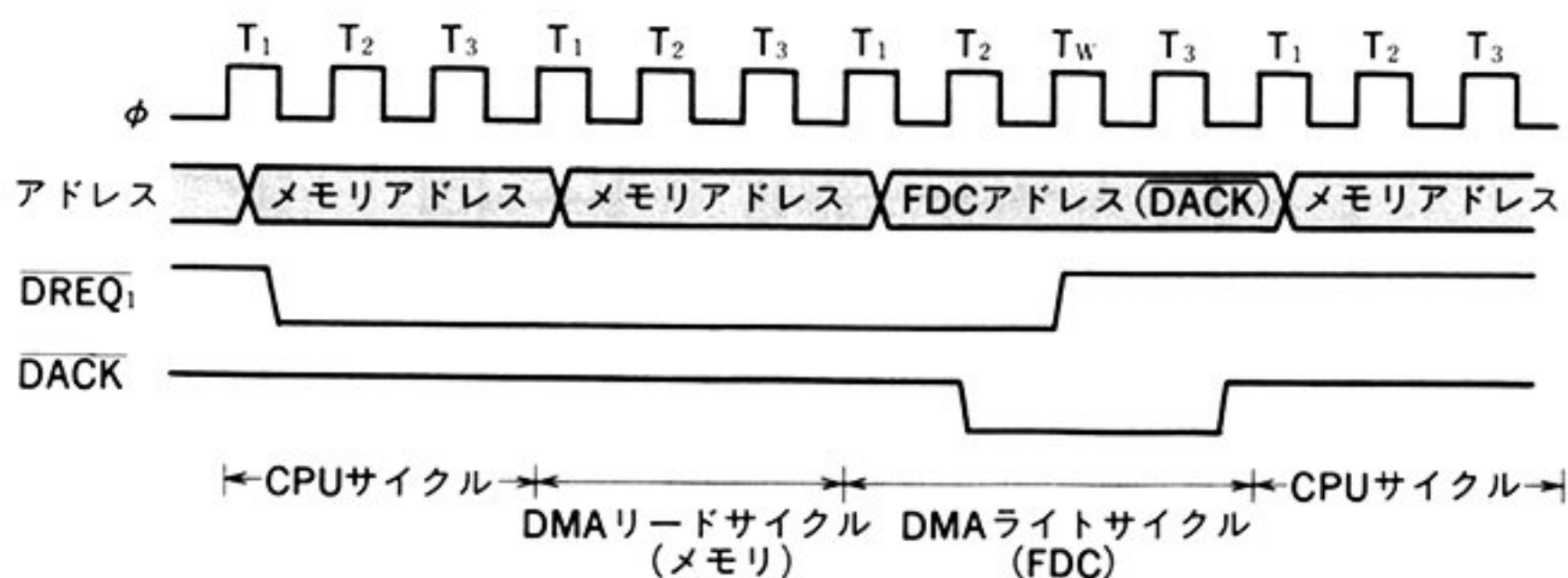



図 13・7　メモリ→FDCデータ転送タイミング

13・5 CRTCの接続

64180を使用してCP/Mシステムなどのパソコンやワープロを構築する際必要になるのが，CRTコントローラ（以下CRTCと略す）やLCDタイミングコントローラ（以下LCTCと略す）などの表示系コントローラの制御です．HD6445はCRT表示に必要な各種タイミング信号を発生し，画面分割やスムーズスクロールなどの各種機能を備えた，小形の簡易キャラクタディスプレイから高解像度のフルグラフィックシステムまで適用できるCRTCです．**図13•8**に，HD6445を使用したキャラクタ表示システムを示します．

64180とHD6445を接続する場合，$\overline{\text{CS}}$信号はアドレスのみから生成します．

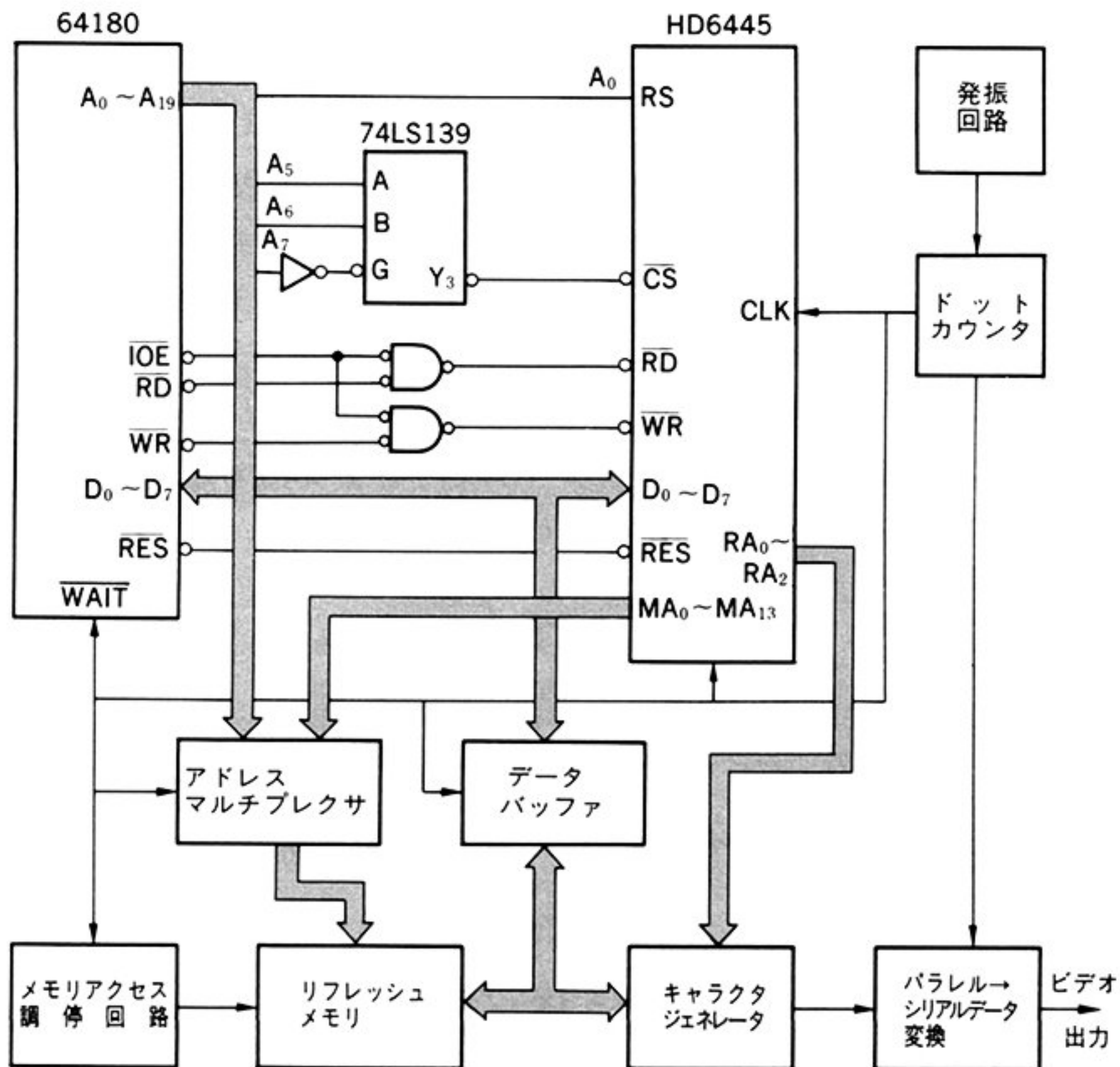



図 13・8 HD6445によるCRT表示システム構成例

HD6445 の $\overline{\mathrm{RD}}$，$\overline{\mathrm{WR}}$ 信号は 64180 の $\overline{\mathrm{IOE}}$ 信号と $\overline{\mathrm{RD}}$，$\overline{\mathrm{WR}}$ 信号の AND をとって生成します．HD6445 は表示タイミングのみ出力するため，64180 が表示データをリフレッシュメモリにライトし，その表示データを HD6445 が CRT に出力する形をとります．リフレッシュメモリを 64180 と HD6445 の 2 つの LSI がアクセスするため，バスの競合がおこります．バスの競合を回避するため，64180 と HD6445 の出力アドレスを切り替えるアドレスマルチプレクサ回路，データバスを切り替えるデータバッファ，およびアクセスタイミングを調整するメモリアクセス調停回路が必要になります．

リフレッシュメモリへのアクセスを CPU 優先にすると，CRTC アクセスが阻害されるため，CRT 上にちらつきが出ます．ちらつきを防止するためには，CRTC のアクセスを優先する必要があります．CRTC アクセスを優先するために，CRTC がリフレッシュメモリをアクセスしている間，CPU がリフレッシュメモリをアクセスしないように，ハードウェアまたはソフトウェアで対処します．もともと非同期である CRTC と CPU のバスタイミングを，CPU がリフレッシュメモリをアクセスするときのみ 64180 の $\overline{\mathrm{WAIT}}$ 信号を制御して，CRTC のバスタイミングにあわせます．この方法はメモリアクセス調停回路が複雑になりますが，プログラム実行のスループットが上がります．**図 13・9** にアクセスタイミングを示します．

高解像度グラフィックシステム，たとえば 640×400 ドット RGB カラー表示システムでは，96 K バイトのリフレッシュメモリが必要ですが，64180 内蔵の MMU を使うことで簡単にメモリ管理が行えます．

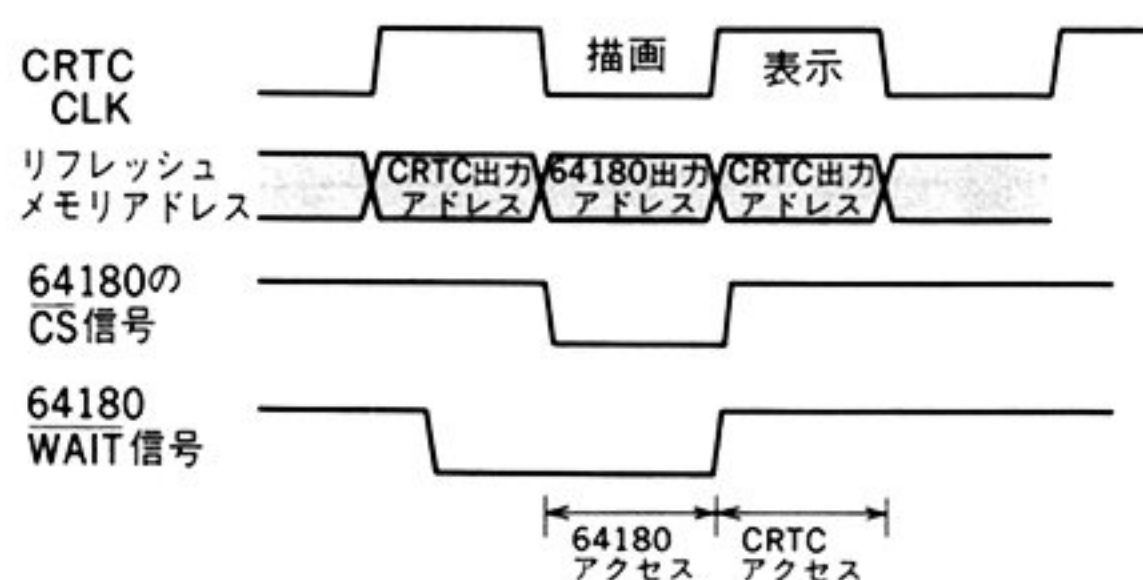



図 13・9　リフレッシュメモリアクセスタイミング

13・6 LCTCの接続

13・5 節ではCRTCを接続しましたが，本節ではLCDタイミングコントローラ（以下LCTCと略す）を接続した例を示します（**図13・10**）．HD64645は，大容

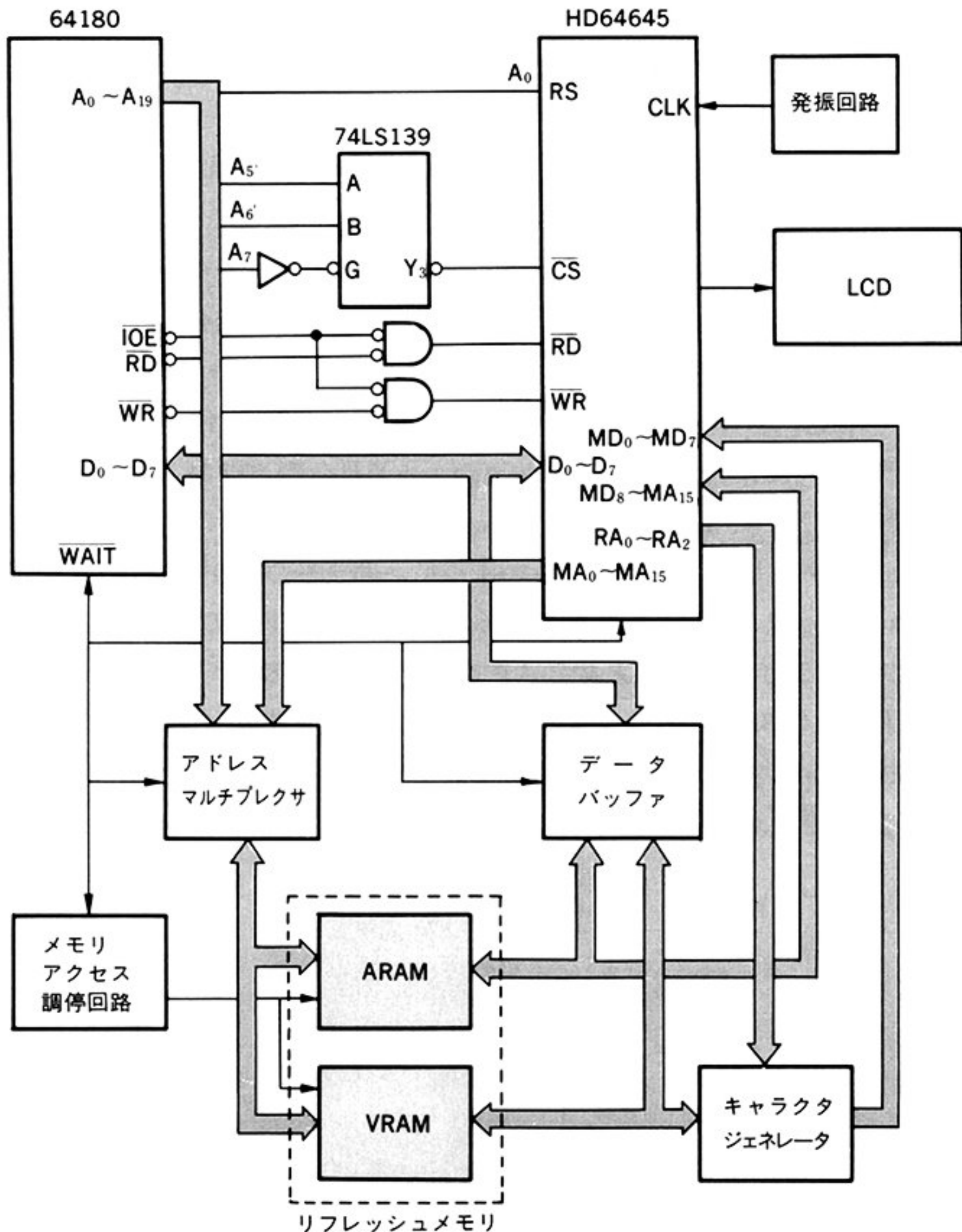



図 13・10 HD64645によるLCD表示システム構成例

量ドットマトリクス液晶表示コントローラで，6845CRTCとソフトウェアコンパチブルであり，さらにスムーズスクロール，キャラクタ単位の反転ブリンク，キャラクタ・グラフィック重合せ表示用のアトリビュート機能をもっています．そのため，6845を使用したCRT表示システムをLCD表示システムに置き替える場合，ソフトの遺産を活用できます．

HD64645によるLCD表示システムは，基本的にHD6445によるCRT表示システムと同様です．64180とHD64645のバスの競合がおきないように，64180とHD64645の出力アドレスを切り替えるアドレスマルチプレクサ回路，データバスを切り替えるデータバッファ，アクセスタイミングを調整するメモリアクセス調停回路が必要です．HD64645は，リフレッシュメモリを2バイト単位で管理しています．たとえば，キャラクタ表示モードのときはキャラクタデータ用VRAMとアトリビュート用ARAMの2バイト構成で，HD64645は2バイト同時にアクセスします．64180はリフレッシュメモリを1バイト単位でアクセスするため，VRAMとARAMのアドレスを違える必要があります．

また，CRTCの場合と同様に，リフレッシュメモリへのアクセスをLCTC優先にする必要があります．LCTCがリフレッシュメモリをアクセスしている間，CPUがリフレッシュメモリをアクセスしないように，ハードウェアまたはソフトウェアで対処します．もともと非同期である，LCTCとCPUのバスタイミングをCPUがリフレッシュメモリをアクセスするときのみ64180の$\overline{\text{WAIT}}$信号を制御して，LCTCのバスタイミングにあわせます．この方法は，メモリアクセス調停回路が複雑になりますが，少しの待ち時間でリフレッシュメモリをアクセスできるため，プログラム実行のスループットが上がります．

14. 新規命令

64180はZ80の158個の命令に加え，7種類12個の命令が追加され165個の命令をもっています．これらの命令は，64180に内蔵された周辺機能を効率よく使用したり，Z80に比べ大幅にスループットを向上させるうえで不可欠な命令です．また，スリープ命令のようにCMOSの低消費電力を生かすうえで，重要な命令も含まれています．これらの命令を使用することにより，効率よいプログラムを作成することができます．

14・1　新規命令の概要

64180は，Z80と互換性のある命令セット（158命令）に加えて，7種類12個の命令が追加され165個から成る命令セットをもっています．（**表14・1**）．

表 14・1　追加命令一覧

	ニーモニック	区　分	機　能
1	INO g, (m)	I/O命令	$(OOm)_I \to g$ $m \to A_0 \sim A_7$ $OO \to A_8 \sim A_{15}$
2	OUTO (m), g	I/O命令	$g \to (OOm)_I$ $m \to A_0 \sim A_7$ $OO \to A_8 \sim A_{15}$
3	OTIM	I/O命令	$(HL)_M \to (OOC)_I$ $HL+1 \to HL$ $C+1 \to C$　$C \to A_0 \sim A_7$ $B-1 \to B$　$OO \to A_8 \sim A_{15}$
	OTIMR		Q [$(HL)_M \to (OOC)_I$ $HL+1 \to HL$ $C+1 \to C$　$C \to A_0 \sim A_7$ $B-1 \to B$　$OO \to A_8 \sim A_{15}$] Repeat Q Until B=0
	OTDM		$(HL)_M \to (OOC)_I$ $HL-1 \to HL$ $C-1 \to C$　$C \to A_0 \sim A_7$ $B-1 \to C$　$OO \to A_8 \sim A_{15}$
	OTDMR		Q [$(HL)_M \to (OOC)_I$ $HL-1 \to HL$ $C-1 \to C$　$C \to A_0 \sim A_7$ $B-1 \to B$　$OO \to A_8 \sim A_{15}$] Repeat Q Until B=0
4	MLT ww	8ビット算術命令	$WWHr \times WWLr \to WW_R$
5	TSTIO m	8ビット論理演算命令	$(OOC)_I \cdot m$　$C \to A_0 \sim A_7$ $OO \to A_8 \sim A_{15}$
6	TST g	8ビット論理演算命令	$A_{cc} \cdot g$
	TST (HL)		$A_{cc} \cdot (HL)_M$
	TST m		$A_{cc} \cdot m$
7	SLP	CPUコントロール	Sleep

この追加された12命令は，次の3種類に大別することができます．

(1) 64180の内蔵する周辺機能を効率よく使用するための入出力命令

　IN0，OUT0 命令など

(2) Z80に備えていない命令で処理能力を向上する命令

　① 論理演算命令，**TST 命令**など

　② 乗算命令（**MLT 命令**）

(3) CMOSの低消費電力を有効に活用するための命令（**SLP 命令**）

(1)のIN0，OUT0などの命令は，内蔵の周辺機能を使用する場合に便利です．内蔵の周辺機能のレジスタは，64180のもつ64Kバイトの I/O 空間中の64バイトに割り当てられており，IN0，OUT0などの命令は，この周辺機能のレジスタを効率よくアクセスするための入出力命令です．

(2)のTST命令は，RAM上のフラグをチェックする場合などに使用します．TST命令はAND命令とは異なり，RAMや I/O レジスタの値に影響を与えることなくフラグなどのチェックが可能です．また，MLT命令は制御応用には必須の機能で，8ビット×8ビットの乗算命令です．(2)のTST命令，MLT命令はいずれもZ80がもっていない機能を命令としてもち，実行ステートの低減を実現する命令です．

(3)は低消費電力モード（スリープモード）を設定する命令です．SLP命令は，64180のようなCMOSのマイクロプロセッサには必須の命令といえます．

また，これらの拡張命令は，マクロ80のような従来，8085，Z80用として広く普及しているマクロアセンブラを用いて使用することができます．具体的には，拡張された命令をマクロ定義することにより，Z80用のマクロアセンブラを64180用のアセンブラとして使用することができます．

14・2　入出力命令の使用例

64180は内蔵の周辺機能を効率よく使用するため，入出力命令（I/O命令）を追加しています．追加された命令は，次に示す3種類6つの命令です．

(1)　INO命令

(2)　OUTO命令

(3)　OTIM，OTIMR，OTDM，OTDMR命令

内蔵周辺機能のレジスタは，64Kバイトの I/O 空間上の0番地から256番地の中に配置されています．実際には，この中の64バイトだけが使用されています．Z80の命令を用いてこれらのレジスタをアクセスしようとする場合，ロード命令でBCレジスタに16ビットのアドレスを設定したうえで，OUT（C），g命令などの命令を使用します．従来，Z80でI/Oの書込みに使用したOUT（m），Aは，アキュムレータの内蔵がアドレスの上位（A_8～A_{15}）に出力されるため，内蔵周辺のレジスタ書込みには使用できません（図14・1）．

これに対し(1)～(3)の命令は，BCレジスタへの設定をせずに周辺機能のレジスタをアクセス可能です．(1)～(3)の追加命令は，オペランドで下位8ビット（A_7～A_0）を設定し，上位8ビット（A_{15}～A_8）には自動的に00Hが設定されます．

たとえば，内蔵のDMACのレジスタを設定する例を示します．DMACは設定すべきレジスタが多く，効率よくレジスタの設定を行う必要があります．Z80では，OUTI，OUT（C），A命令を使用しています．これに対し64180では，OTIMR，OUTO命令を使用し，レジスタの設定を行っています．ここでOTIMR

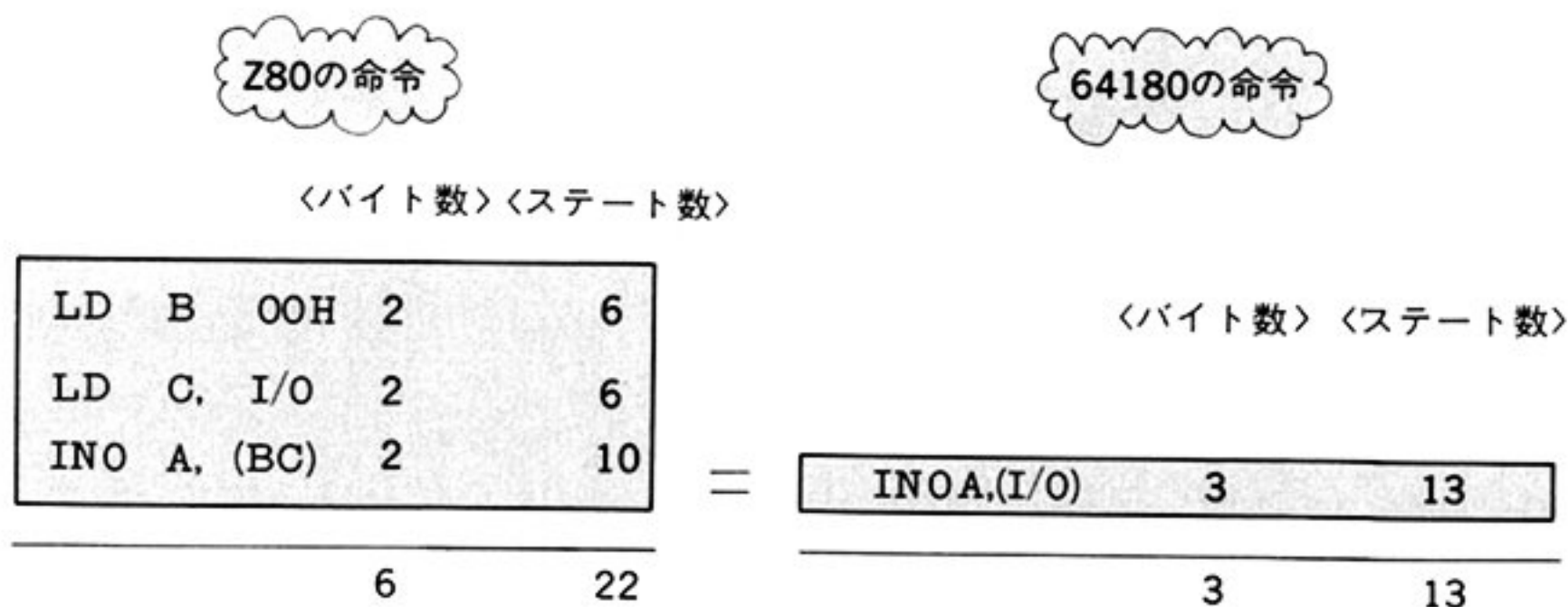



図14・1　I/O空間上のレジスタ設定を行うプログラム例

Z80の命令を使用した場合			〈バイト数〉	〈ステート数〉	64180の命令を使用した場合		〈バイト数〉	〈ステート数〉
	LD	HL,1000H	3	9	LD	HL,1000H	3	9
	LD	BC,0020H	3	9	LD	BC,0820H	3	9
	LD	D,08H	2	6				
LOOP	OUTI		2	12	OTIMR		2	16 不成立 14 成　立 SAR DARの設定 BCR
	INC	B	1	4				
	INC	C	1	4				
	DEC	D	1	4				
	JR	NZ,LOOP	2	6不成立 8成　立				
	LD	A,C3H	2	6	LD	A,C3H	2	6
	LD	C,31H	2	6				
	OUT	(C),A	3	10	OUTO	A,31H	2	13 DMAモード レジスタの設定
	LD	A,66H	2	6	LD	A,66H	2	6
	DEC	C	1	4				
	OUT	(C),A	3	10	OUTO	A,30H	2	13 DMAステータス レジスタの設定 (DMAイネーブル)
14行			28	320	7行		16	182

図 14・2　追加された入出力命令の使用例

命令はI/Oへのブロック転送命令で，メモリから連続するI/Oへのブロック転送を行います．この例では，OUTO，OTIMR命令の使用により，プログラムライン数，バイト数，実行ステート数とも，Z80の命令を使用した場合に比べ50～60%に削減しています（**図14・2**）．

なお，このプログラムはチャネル0のDMACで，メモリ↔メモリの転送をサイクルスチールモードで実行させる例です．

14・3 論理演算命令（TSTIO命令）の使用例

TSTIO命令も，主に64180の内蔵する周辺機能を考慮して追加された命令です．TSTIO命令はCレジスタに下位8ビットのI/Oアドレス（A_7～A_0）を指定すると上位8ビット（A_{15}～A_8）に'00'が指定され，このアドレスのレジスタの値とオペランドで指定した8ビットのデータとの論理積をとる命令です．結果はフラグにのみ反映される非破壊の論理演算を行います（**図14・3**，**表14・2**）．

Z80の命令を使用する場合，IN A,(C)などの入力命令により，アキュムレータにI/Oのレジスタを読み出し，論理演算命令（AND m など）により'1'，'0'の判定を行います．64180では，この動作をTSTIO命令，1命令で実現することができます．またAND命令の場合，1度論理演算を実行するとその値が破壊されるため，再度論理演算を行う場合，もう1度I/Oから読み出すか他のレジスタに退避しておく必要があります．これに対して，TSTIO命令は非破壊であるため，何度でも異なる値との論理演算を実行することができます．

たとえば，TSTIO命令を用いて内蔵の非同期シリアル（ASCI）のフラグを

図14・3 TSTIO命令の等価機能

表14・2 TSTIO命令とAND命令の実行結果

	フラグ	レジスタへの反映
TSTIO 命令	変化する	反映しない（非破壊）
AND命令	変化する	A_{cc} に反映する（破壊）

Z80の命令を使用した場合	〈バイト数〉	〈ステート数〉	64180の命令を使用した場合	〈バイト数〉	〈ステート数〉	
LD C,04H	2	6	LD C 04H	2	6	
LD B,00H	2	6				
IN A,(C)	2	9				
LD B,A	1	4				
LD A,74H	2	6				
AND B	1	4	TSTIO 74H	3	12	RDRFのみ1
JR Z,L01	2	6不成立 8成立	JR Z,L01	2	6不成立 8成立	
LD A,34H	2	6				
AND B	1	4	TSTIO 34H	3	12	RDRF OVRN =1
JR Z,Lφ2	2	6不成立 8成立	JR Z,L02	2	6不成立 8成立	
LD A,04H	2	6				
AND B	1	4	TSTIO 04H	3	12	RDRF(PE+FE)=1
JR Z,L03	2	6不成立 8成立	JR Z,L03	2	6不成立 8成立	
JR DCD	2	8	JR DCD	2	8	
14行	24	81	8行	19	68	

図 14・4 TSTIO命令の使用例

判定し，フラグに応じた処理を行うプログラムを考えてみましょう．非同期シリアルのステータスレジスタ（チャネル0はI/O空間の04H，チャネル1は05H）には，RDRF，OVRNなど5つ（チャネル0の場合DCDも加え6つ）のステータスフラグがあります．ここでは受信のみについて考慮すると，RDRF，OVRN，PE，FE，DCDの計5つのフラグを判定する必要があります．64180はZ80に比べ，TSTIO命令により効率よくフラグの判定を行えるため，Z80の命令のみを使用した場合に比べ，ライン数で60％，バイト数，実行サイクル数でも80％程度にプログラムを削減することができます（**図14•4**）．

なお，ここに示したプログラムでは，簡単にするためパリティーエラー（PE）とフレーミングエラー（FE）に関する処理は，同一のプログラム（プログラム中のL03の処理）としてあります．

14・4　MLT命令の使用例

64180は，8ビット×8ビットの乗算命令をもっています．この乗算命令は，BCレジスタなどのレジスタペアを使用して8ビット×8ビットの乗算を行います．たとえば，Bレジスタ，Cレジスタにそれぞれ乗数，被乗数を設定しMLT命令を実行すると，積（答え）がBCのレジスタペアにストアされます．この16ビットのペアレジスタとしては，BCレジスタ，DEレジスタ，HLレジスタ，スタックポインタ（SP）の4種類のレジスタを使用できます．MLT命令の実行には17ステートかかり，実際の実行時間は8 MHz版で2.1 μsかかります．

従来，Z80では乗算命令をもっていなかったため，シフト命令を使用して乗算のプログラムを作成する必要がありました．これに対し64180は，MLT命令に

Z80の命令を使用した場合

ラベル	命令	〈バイト数〉	〈ステート数〉
	LD B, 08H	2	6
	LD A, (RAMB)	3	12
	LD(RSLT+1), A	3	13
	XOR A	1	4
	LD(RSLT), A	3	13
LOO	LD HL, RSLT	3	9
	SLA(HL)	2	13
	INC HL	1	4
	RL(HL)	2	13
	JR NC LO1	2	6 不成立 8 成立
	LD HL, (RAMA)	3	15
	LD DE, (RSLT)	4	18
	ADD HL, DE	1	7
	LD(RSLT), HL	3	16
LO1	DEC B	1	4
	JR NZ, LOO	2	6 不成立 8 成立
16行		36	938

64180の命令を使用した場合

命令	〈バイト数〉	〈ステート数〉
LD A, (RAMB)	3	12
LD H, A	1	4
LD A, (RAMA)	3	12
LD L, A	1	4
MLT HL	2	17
LD(RSLT), HL	3	16
6行	15	65

図 14・5　MLT命令の使用例

表 14・3 追加命令によるスループットの向上

	ライン数		64180/Z80	バイト数		64180/Z80	ステート数		64180/Z80
	Z80の命令	64180の命令		Z80の命令	64180の命令		Z80の命令	64180の命令	
入出力命令	14	7	0.5	28	16	0.62	320	182	0.75
TSTIO命令	14	8	0.57	24	19	0.79	81	68	0.84
MLT 命令	16	6	0.38	36	15	0.42	938	65	0.07
計	44	21	0.48	88	50	0.57	1339	315	0.24

より大幅にスループットの向上を実現することができます（**図14・5**，**表14・3**）．

たとえば，8ビット×8ビットの乗算を実行しようとした場合，Z80では，シフト命令を使用しループを構成します．

一方，64180ではこれをMLT命令で実現しています．Z80と64180の実際のプログラムを比較すると，64180ではMLT命令の使用によりプログラムのライン数，バイト数では40％程度，実行ステート数では10％程度にプログラムを削減しています（**図14・6**）．

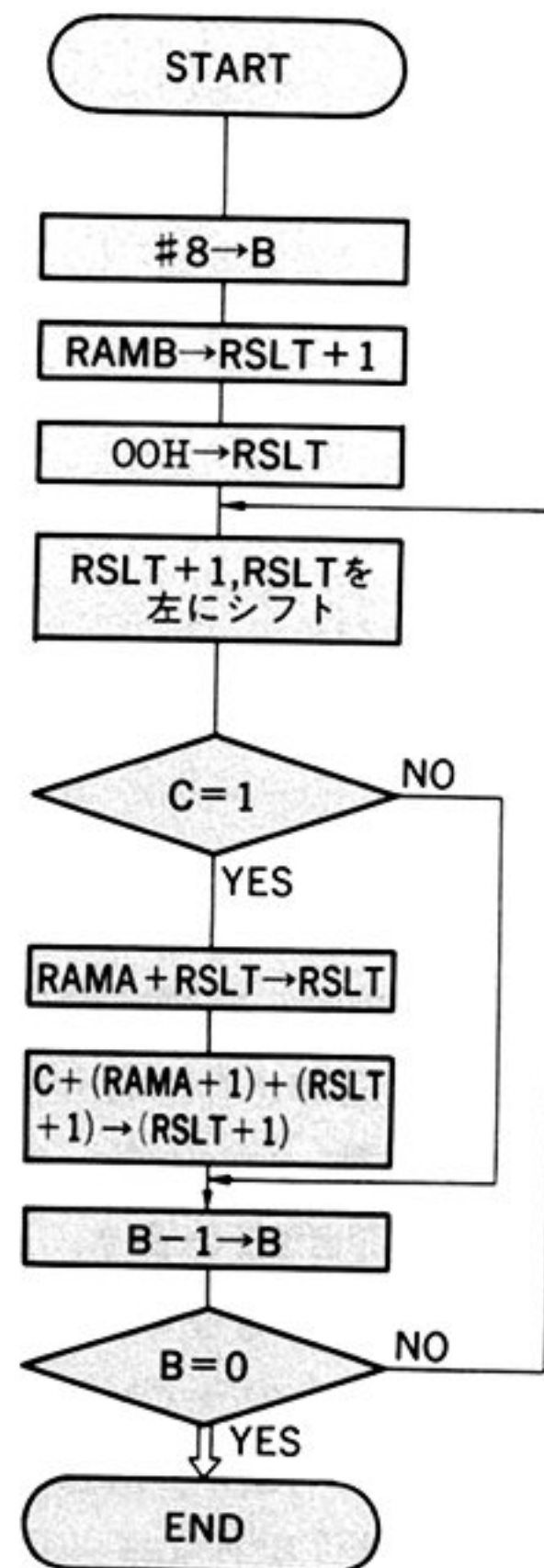



図 14・6 Z80の命令を使用した場合の乗算-フローチャート

拡張命令によるスループットの向上

64180では追加された命令を使用することにより，Z80に比べ大幅にプログラムのライン数，バイト数，実行ステート数を削減することができます．表に入出力命令，TSTIO命令，MLT命令の使用効果をまとめてみます．表14・3はZ80と64180の命令で，入出力，フラグ判定，および8ビット×8ビットの乗算を行った場合のライン数，バイト数，ステート数およびZ80と64180の性能比を示してあります．64180はZ80に比べ，ライン数で48％程度，バイト数で57％程度，実行ステート数で24％程度にプログラムを削減し，スループットの向上を実現できることがわかります．

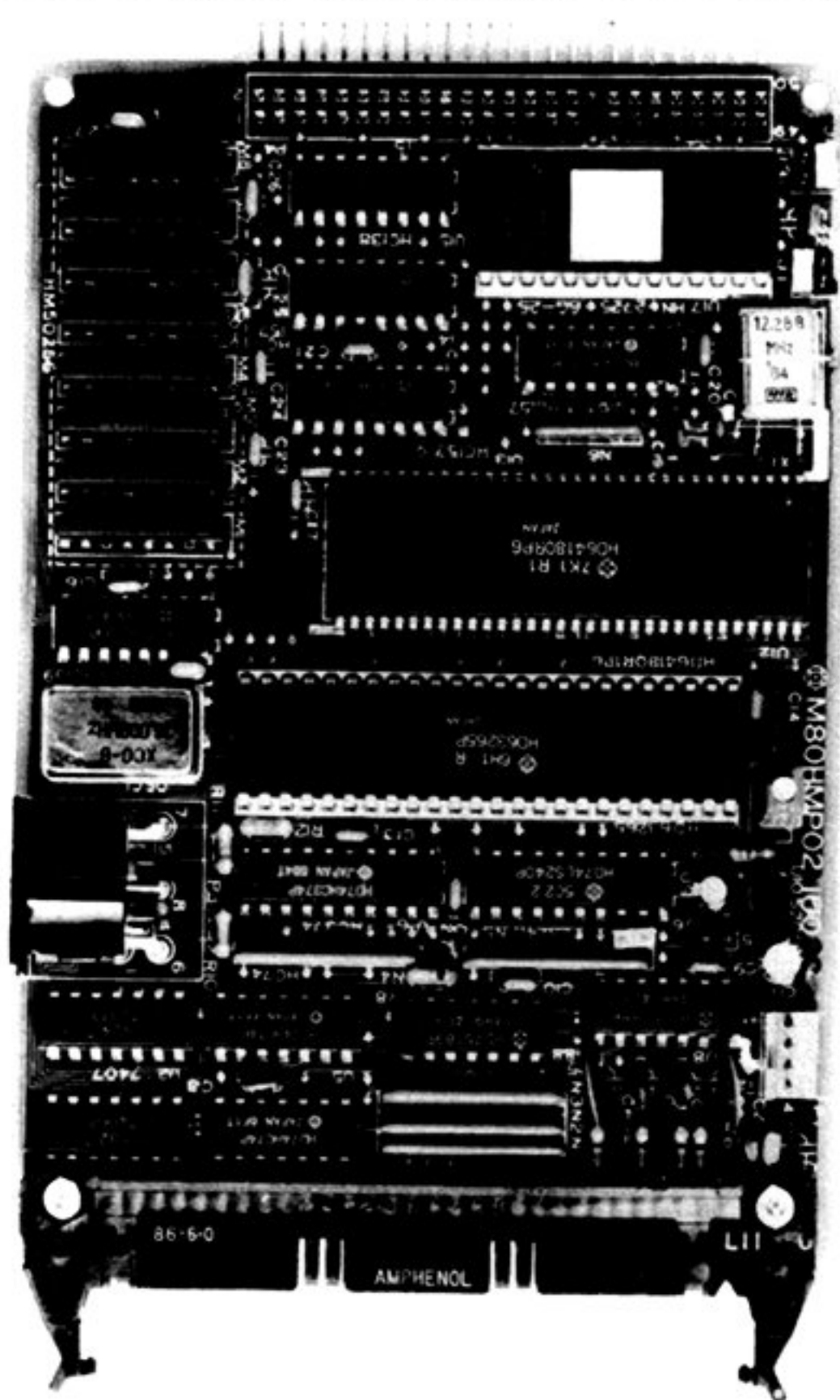
15. 180 カード パソコン

カードパソコンは 80 系ソフトウェア資源を生かしつつ，小形でコンパクトなパソコン機能をもつシステムです．ボードの構成は，80 系の汎用 OS である CP/M Plus が走る最小構成をとっており，CP/M の応用プログラムによるソフトウェア開発装置としても利用できます．主な構成は，メモリ（RAM：256 K バイト，ROM：32 K バイト），RS-232C 通信インタフェース，プリンタインタフェースおよびフロッピーディスクインタフェースから成り，ハガキサイズ（148×100 mm）の大きさで実現しています．

15・1 カードパソコンについて

64180を搭載するシステムを構築する場合

(1) Z80とその周辺LSIを用いたシステムの小形化が図れる

(2) メモリ空間が広くとれ（最大1Mバイト），80系オペレーティングシステム（OS）との組合せで高級言語など幅広いソフトウェアの活用ができる

ROMゴースト

ROMの選択は64180のすべてのアドレスについてデコードをしていないために，見かけ上何もないメモリ空間でもROMが選ばれ，データが読み出される．

図 15・1 外観図

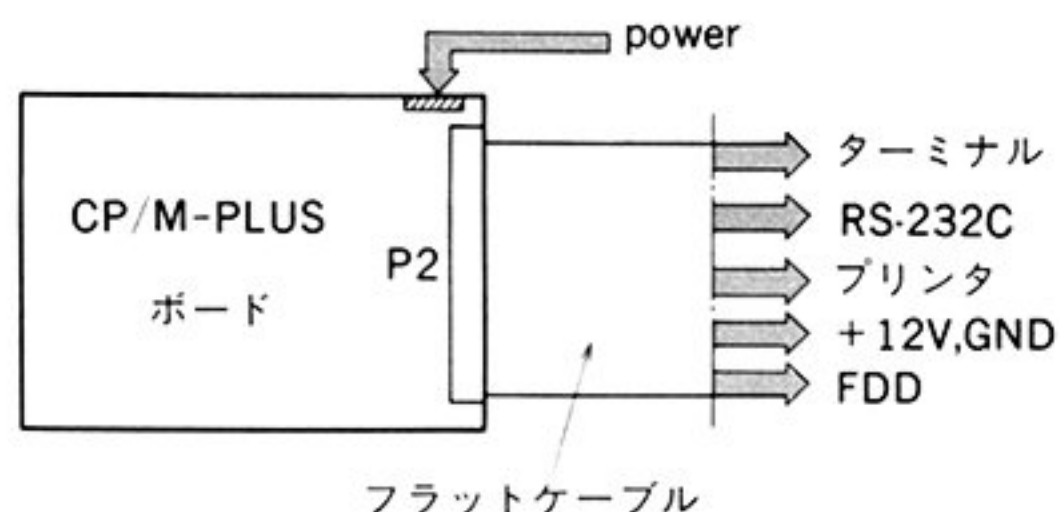



図 15・2　周辺機器接続方法

などの特長を出すことができます．

カードパソコンでは，これらの特長をもつパソコンシステムを実現しています．設計上のコンセプトとして，パソコンをハガキサイズ（148×100 mm）の基板に1ボード化することをハードウェアの主眼とし，さらに膨大な80系ソフトウェア資産を生かせるように，80系標準OSである米国ディジタル・リサーチ社のCP/Mを搭載することをソフトウェア上のポイントとしています．特に，CP/M-80の最新バージョンであるCP/M3.0（＝CP/M-Plus）はバンク機能をもち，64180内蔵MMUと組み合わせることによりプログラムエリアを拡張できるもので，C言語，BASIC言語，FORTRANなどCP/M応用プログラムを使うプログラム開発では必須です（**図15・1**）．

さらに，CP/Mにはマクロアセンブラ（MACRO-80）などの基本プログラムもあり，このシステムだけで開発装置としても活用できるとともに，実際に64180レベルでの評価を行うことにも利用できます．

システムをインプリメントするにあたっては，64180内蔵のI/O以外の外部I/Oを最小とすることを目標とし，以下の2つのI/Oだけとしています．

- フロッピーディスクインタフェース
- プリンタインタフェース

フロッピーディスクインタフェースは，CP/Mがディスクオペレーティングシステムであることから必要です．フロッピーディスクのデータ転送速度は250 KBPSで，PIOモードの転送でも可能ですが，MPUのスループットを向上させるため64180のDMACと組み合わせて用います（DMA転送）．

プリンタインタフェースはパソコン機能の1つとしてもたせていて，これらのI/Oインタフェースは外部コネクタにより周辺機器と接続します（**図15・2**）．

15・2 システム構成

〔1〕 **I/O の 構 成**　システム構成は，15・1節で述べたI/O以外にキー入力と表示出力のために64180のASCIがあり，ASCIの他方（ch0）は他のシステムとの通信用として用いています．いずれもRS-232Cインタフェース仕様としてドライバ/レシーバICを用いています．各部の構成について以下に述べます（**図15・3**）．

(1) フロッピーディスクインタフェース：データセパレータ（VFO回路）を内蔵して，外付部品が少ないHD63265を用います．データ転送はDMA転送とします．HD63265は従来のFDCに対してDMA転送要求からのデータ転送が速く，64180を用いた場合に要求をサンプルしてからDMA転送サイクルを遅らせる必要がなく，外部回路が不要となります．

(2) プリンタインタフェース：セントロニクス8ビットパラレルインタフェースに準拠しています．データ転送は専用LSIではなく，8ビットラッチを用いて小形化を行うため，ソフトウェアでサポートします．

(3) RS-232Cシリアルインタフェース：64180のASCIを使います．ドライバ/レシーバICとして75188/189を用いるため，±12V電源が必要です．

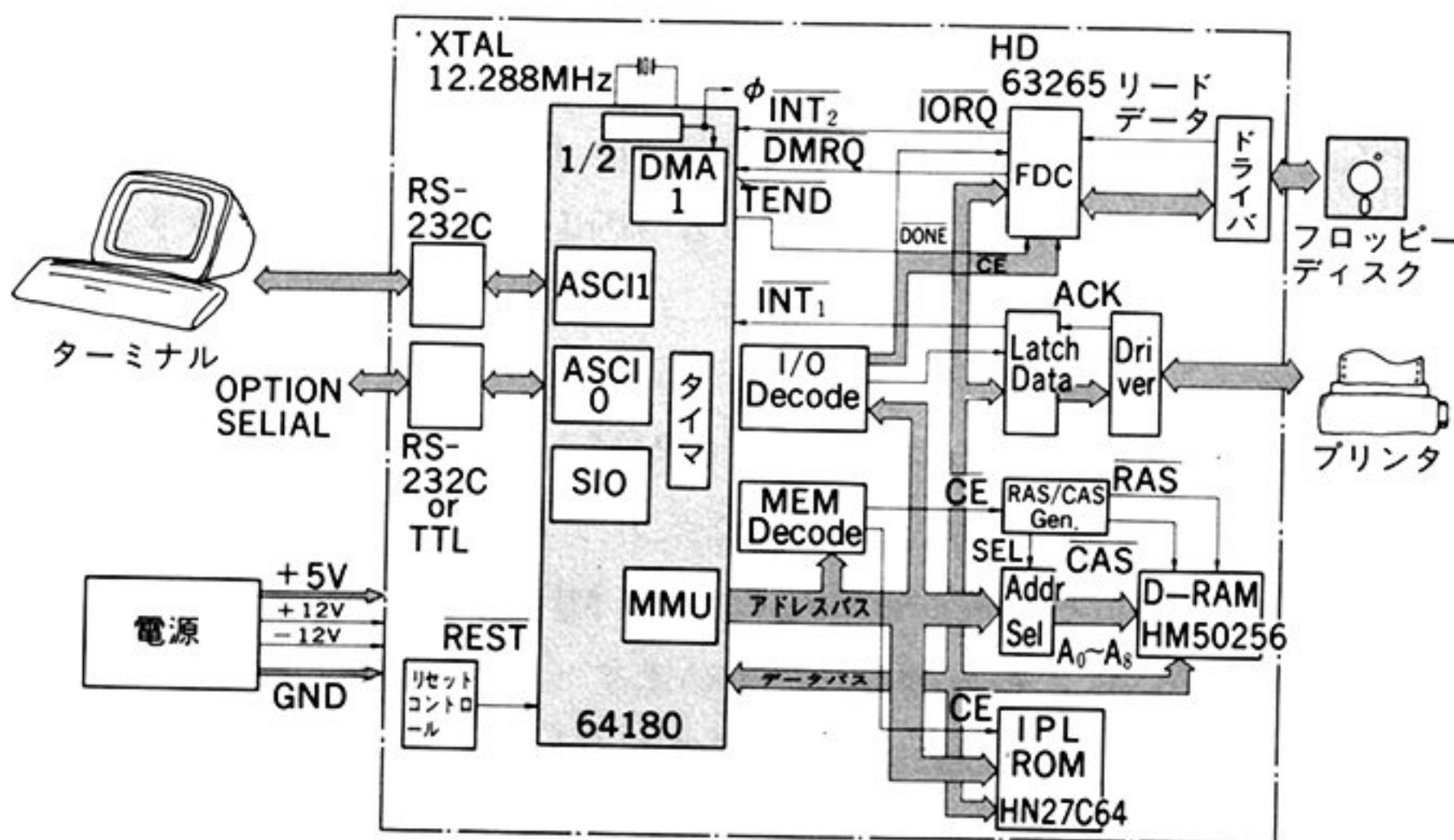



図15・3　ブロック図

〔2〕 メモリ構成　CP/Mを走らせる場合，後述するBIOS，BDOS，CCPなどのメモリ領域の割当てが重要ですが，64180内蔵MMUを用いることによって，物理アドレスに割りつけているメモリを任意の論理アドレスに割り当てることができるため，物理アドレスに対するメモリ構成を意識する必要がありません．

CP/Mを走らせるためには，以下の2つのメモリ構成とします．

(1) DRAM：DRAMとして，HM50256（256 K×1ビット）を8個用いて256 Kバイトのエリアとします．このエリアは主にロードされたCP/Mが常駐し，TPAと呼ばれる応用プログラム領域として使うほか，RAMディスクとして用います．

(2) ROM：27512を用います．CP/Mをファイルからロードするためのコールドスタートローダプログラムと簡易モニタが入ります．

15・3 I/O メモリインタフェース

(1) I/Oインタフェース：このシステムでは，外部I/OとしてはフロッピーディスクインタフェースとプリンタインタフェースでI/Oサイクルのタイミングは FDC のアクセス時間（t_{DDR}）に依存します．HD63265 のアクセス時間は 140 ns で，64180 のI/Oサイクルは4ステート（(1/6.144 MHz)×4＝651 ns）で，プログラマブルウェイトステートの挿入は不要です．また，13・3節で述べたように，FDC のデータ転送をデュアルアドレス方式としているため，DMA 転送アドレス（$\overline{DACK}$ 信号が発生）を別のI/Oアドレス（A0番地）に割りつけます．

プリンタインタフェースは，データ転送信号（$\overline{STROBE}$ 信号）をソフトウェアで発生させています．手順として，C0番地（I/Oライト）にデータを書き込んだ後，C1番地に同一データを書き込むことにより $\overline{STROBE}$ 信号を生成させます．つまり，アドレス A_0＝0 の状態で“H”，A_0＝1 の状態で“L”となり，I/Oアクセスを3回行うことでプリンタにデータを転送させます（**図15・4**）．

(2) メモリインタフェース：ROM と DRAM があります．ROM インタフェースの $\overline{CE}$ 信号はアドレスのデコードにより作り，また $\overline{OE}$ 信号は $\overline{ME}$ 信号とします．したがって ROM はアドレス信号 A_{18}＝“L” かつ $\overline{ME}$ 信号＝“L” のと

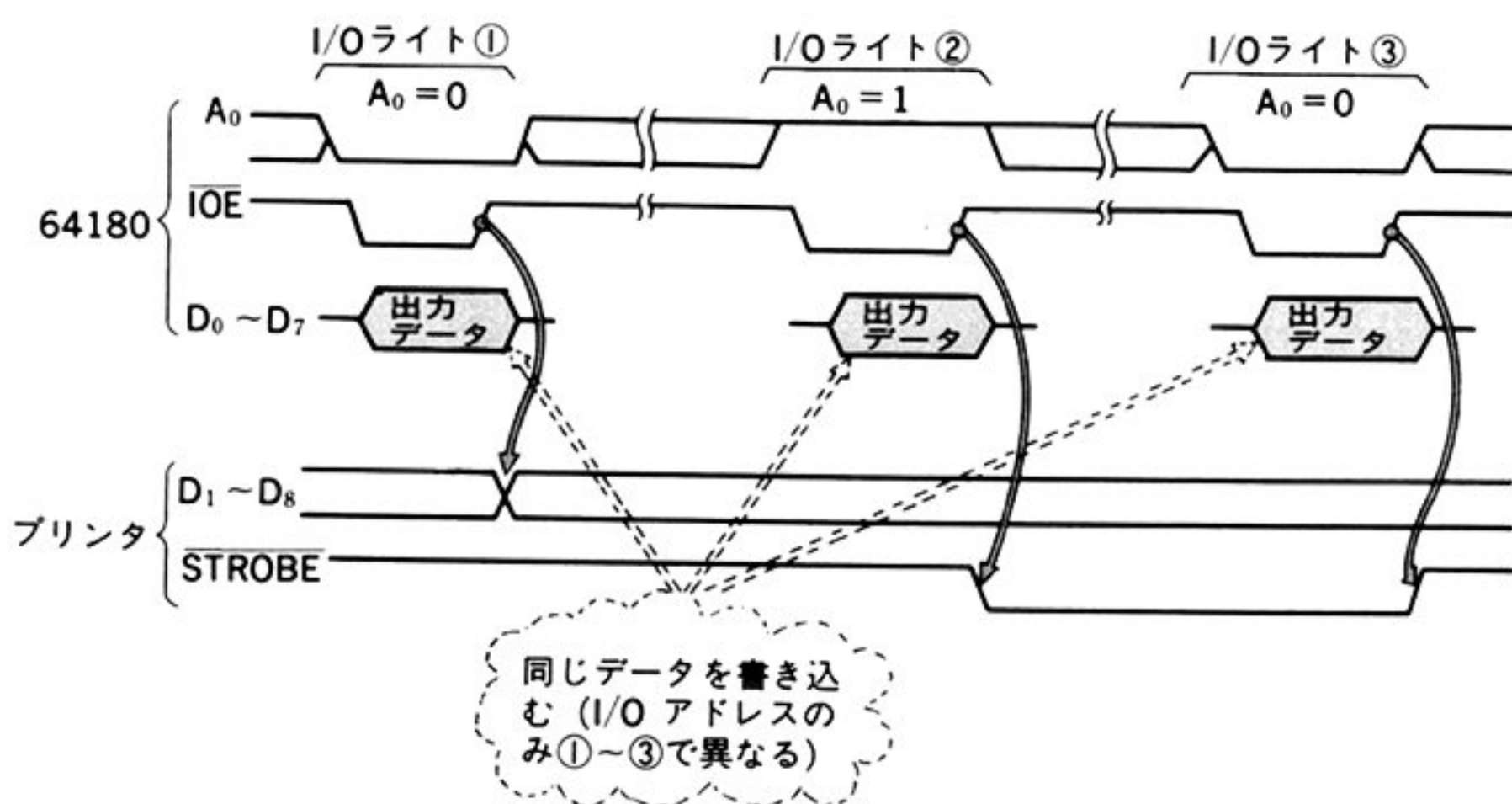



図15・4 プリンタ出力シーケンス

きにアクセスされます．これは回路を省略するために$\overline{RD}$信号と論理をとってなく，メモリライトアクセスではデータがぶつかります．

DRAMインタフェースは13・1節と同様に，Eクロックでロウ/カラムアドレスの切替えタイミングとしているため，$\overline{CAS}$信号を作るのにT_Wステートの挿入が必要となり，メモリアクセスサイクルは，T_1，T_2，T_W，T_3の4ステートとなります．また，DRAMのリフレッシュはRAS-Only-Refreshモードを用いて，4 ms/256行アドレス間隔のリフレッシュを行います．リフレッシュ時には，$\overline{RAS}$信号が発生するだけで$\overline{CAS}$信号は発生しません．リフレッシュ間隔は下式の条件で設定しています（**図15・5**）．

$$\text{リフレッシュ挿入間隔} = N(\text{ステート}) \times \frac{1}{\text{動作周波数}(f)}$$

$$= 80 \times \frac{1}{6.144\,\text{MHz}}$$

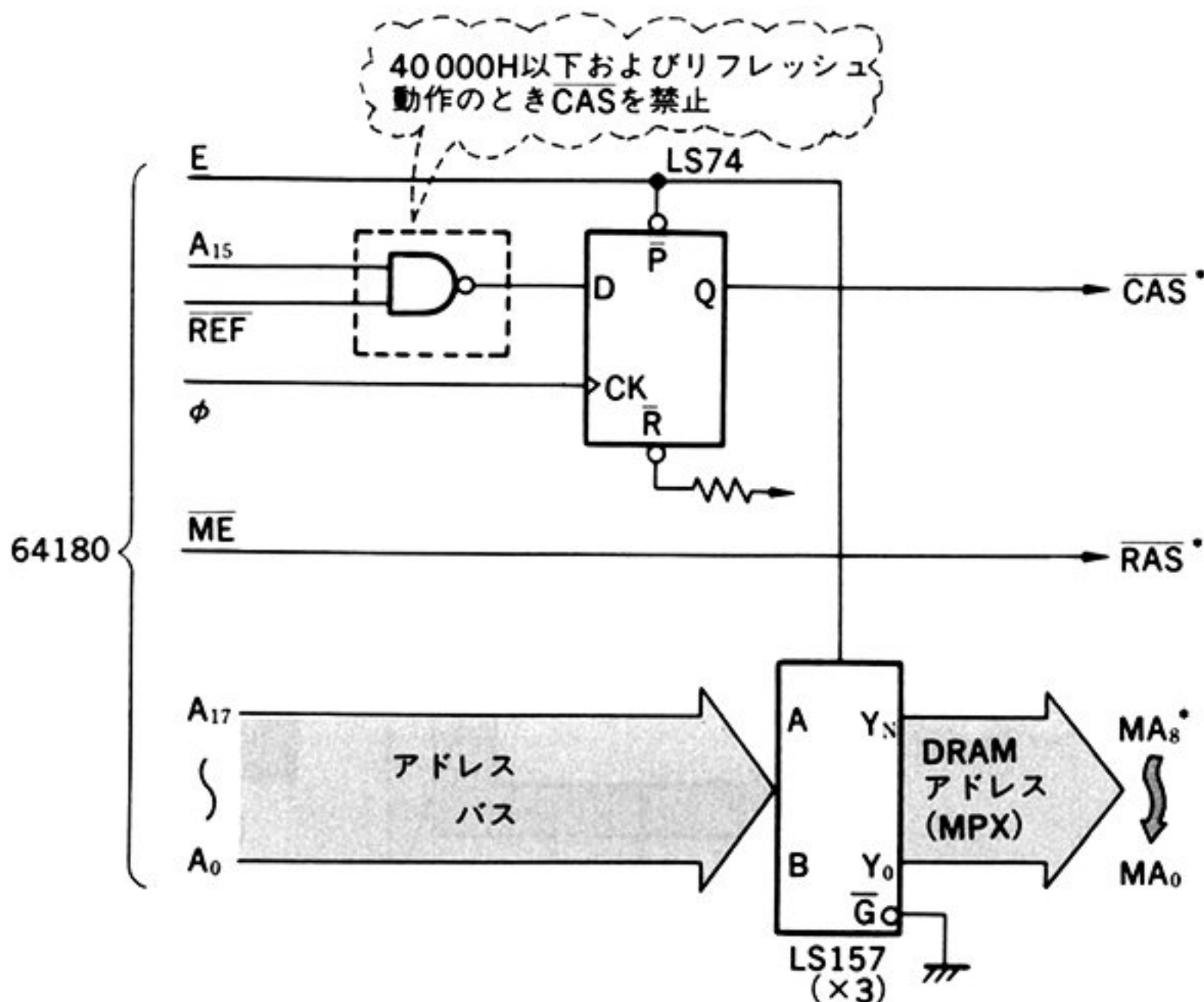
$$= 13.02\,\mu\text{s} < 15.63\,\mu\text{s}\left(= \frac{4\,\text{ms}}{256}\right)$$

注）＊DRAM用のダンピング抵抗は省略してあります．

図 15・5 RAS/CAS回路例

15・4 CP/M-Plus

CP/Mは80系の8ビットマイコンの標準OSとして多用されており，数多くのアプリケーションソフトウェア財産があります．なかでもCP/M-Plusは，8ビットCPU用OSの中で最高の機能をもっており，その大きな特長として次の2つがあります（図15・6）．

(1) バンクメモリシステムによるTPA（ユーザプログラムエリア）の拡大

(2) ディレクトリハッシングとLRUバッファリング技術（ディスクキャッシュ）によるディスクアクセスのスループットの改善

このCP/M-Plusは，汎用OSとして移植性を高めるため，OS全体を次の2つの部分に分けています．

(1) BDOS（Basic Disk Operating System）：CP/M-Plusの中心部分で，主にディスクに格納されているファイルの管理を行っています．また，論理的な周辺装置に対する文字の入出力も行います．

(2) BIOS（Basic I/O System）：文字入出力装置，ディスクドライブ装置などの，周辺装置の制御のようなハードウェアの操作を行います．

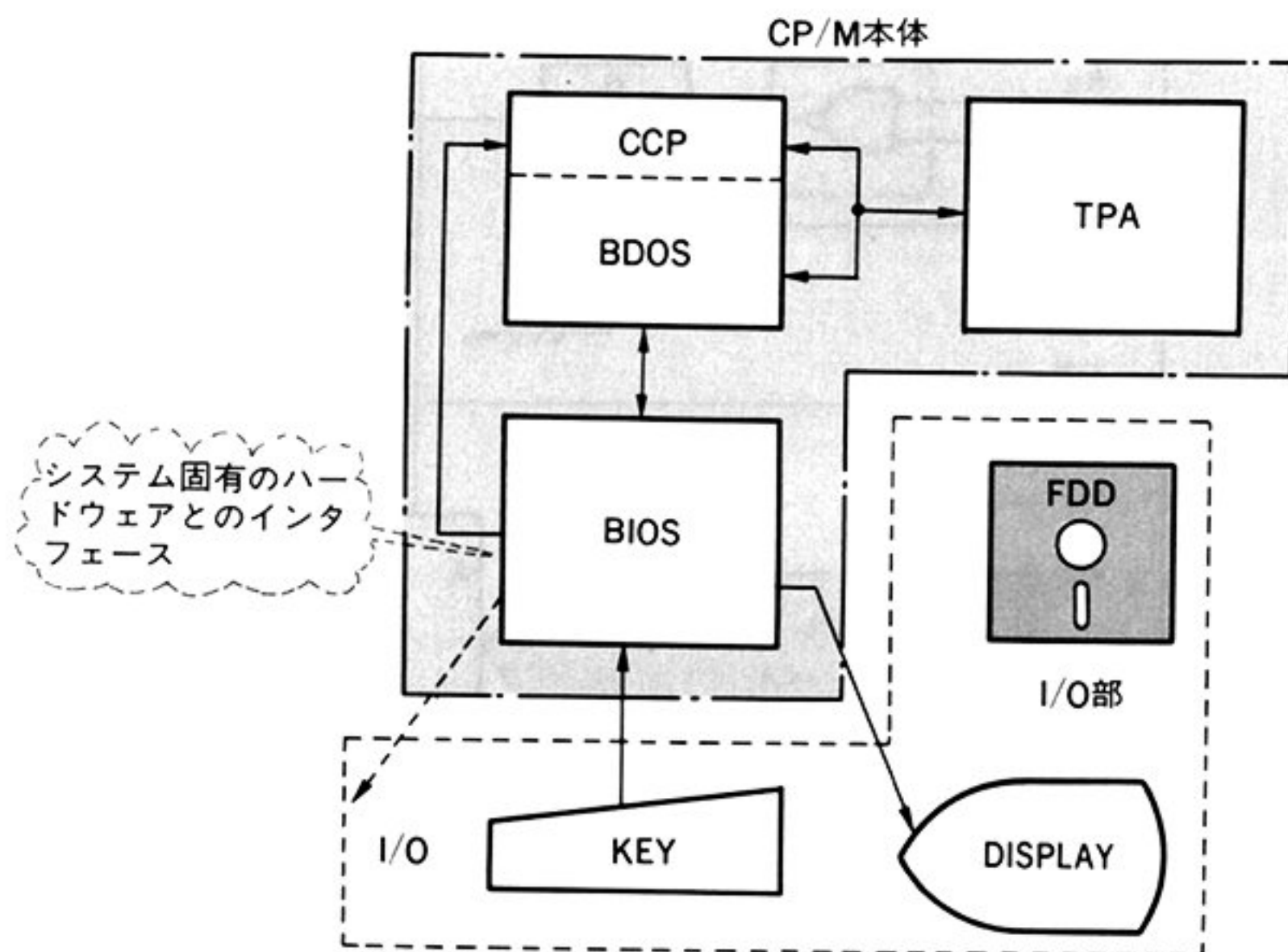



図15・6 CP/Mの構成と処理の流れ

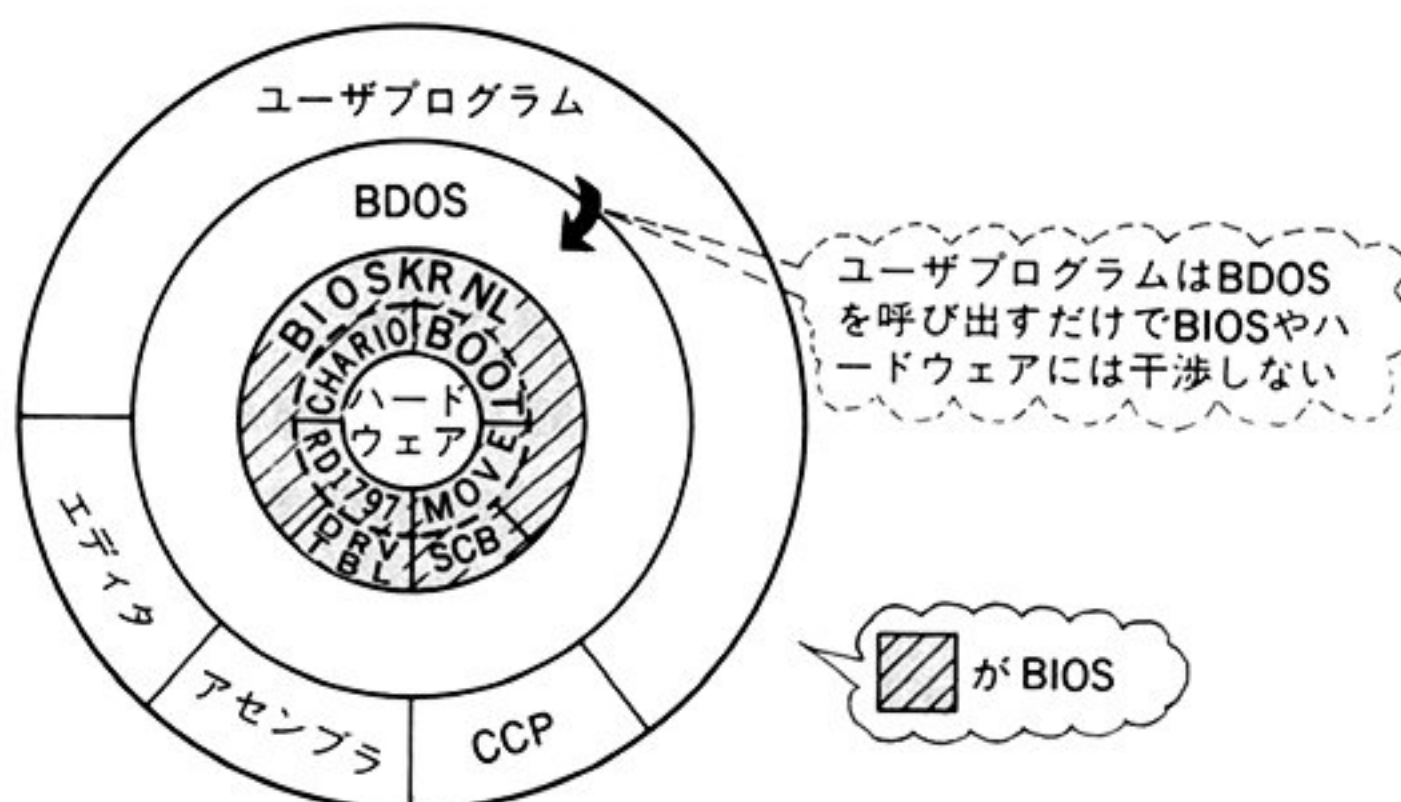



図 15・7 ハードウェア，BIOS，BDOSの関係

以上の2つの部分により，CP/M-Plusシステム全体は層状の構造になり，アセンブラやエディタなどのユーティリティプログラム（ユーザプログラム）はBIOSやハードウェアを操作することなく，BDOSの機能を用いるだけで動作します（**図15・7**）．

64180に搭載されているMMU，DMACは，CP/M-Plusの特長であるバンクメモリ，ディスク，バッファリングを効率的に実現し，CP/M-Plusの能力をさらに引き上げます．しかし，この機能もBIOSの作り方により変わります．64180の内蔵I/Oを活用し，CP/M-Plusの能力を十分に引き出せるようなBIOSを作りましょう．

なお，付録5に180カードパソコンのハードウェア仕様を示しました．また，本書の見返しに回路構成図を示したので参照して下さい．

15・5 BIOSの構成

DRI社から供給されるBIOSのソースプログラムは，主な機能ごとに7つのモジュールに分割されています．この中で，MOVE，CHARIO，DRIVEモジュールをハードウェアにあわせて作成します．これ以外のモジュールは直接ハードウェアに関係することがなく，変更する所も少しだけです．CP/M-Plusの能力を高めるうえで，BIOSの作り方は重要です．64180の機能を使いこなすことが能力を高める大きな要因です．MOVE，CHARIO，DRIVEの3つのモジュールは，以下に示すようにそれぞれ重要なポイントがあります（**表15・1**）．

〔1〕 **MOVE**　　このモジュールにはメモリ↔メモリ間のDMA転送と，バンク切替えの機能があります．バンクの構成方法はCP/M-Plusのメモリマップの

表 15・1　BIOSのモジュール

モジュール名	機　　能
BIOSKRNL	BIOSコールのジャンプテーブルがあり，他のモジュールのルーチンを呼び出すBIOSの中核部分．文字入出力装置やディスクドライブ装置の論理装置－物理装置変換や切替えも行う．移植時にも内容は変更しない．
BOOT	各種文字入出力，ディスクドライブ装置のイニシャライズやCCPコマンドのロード・リロードを行う．このCCPコマンドのリロードは内蔵DMACを使うと高速になる．他にタイマルーチンがあるが内蔵タイマの割込みでタイマのカウントアップを行うときには何もしない．
MOVE	バンクの切替え，メモリデータのブロック転送を行う．特に，ブロック転送はシステムを高速にする．
CHARIO	文字入出力装置のイニシャライズ，入出力動作を行う．カードパソコンではRS-232Cが2チャネルとプリンタ用のチャネルがある．
DRVTBL	CP/M-Plusの論理ドライブであるA～Pと物理ドライブの関係を定義する．このテーブル上のドライブがすべての操作（GENCPMも）の対象となる．
DRIVE (FD1797SD)	ディスクドライブのイニシャライズ，セクタリード/ライトを行う．CP/M-2.2まではこのルーチン内でセクタのブロック/デブロックを行っていたが，CP/M-PlusではこのH作業をBDOSが行っている．
SCB	BDOSとBIOSの共通の変数エリア．主にBIOSの重要な設定値が入っている．

考え方やDMA転送時の問題もあるので，よく考えて決めます．特に，メモリ↔メモリ間のDMA転送はメモリの効率やスピードを決める重要な要素です．

〔2〕 **CHARIO** このモジュールは1文字単位の入出力を行います．カードパソコンでは64180内蔵のASCIが2つとプリンタインタフェースを制御します．ASCIはフラグの状態を見るだけで送受信できますが，プリンタは割込みを使います．このため，CP/M-Plusシステムを高度にするためには，ASCIの割込み機能を使ってリングバッファができるようにします．

〔3〕 **DRIVE** このモジュールではFDCの制御を行います．ここでもFDCからの割込みを使って，コマンドの終了以外にフロッピーディスクドライブのREADY状態のチェックを行います．

以上のBIOSを作り，アセンブル・リンクした後，DRI社から提供されるGENCPMを動作させると，CPM3.SYSが作られます．このようにCP/M-Plusの移植には，少なくともCP/M-2.X以上のマシンを使います．

15・6 MMUとCP/M-Plusのバンク

CP/M-Plusのメモリ空間は，64Kバイトの論理アドレス空間を2つに分け，アドレスのFFFFH側の約4Kバイトを，コモンエリア，0側の約60Kバイトをバンクエリアとして配置しています．バンクエリアは2~16まで置くことができ，ソフトウェアにより指定のバンクエリアと切り替えます．コモンエリアはバンクエリアの切替えに関係なく，各バンクに共通となります．バンク0とバンク1には特別の役割があり，その他のバンクは，ディスクバッファやRAMディスクとして使用できます．バンク0はCP/M-Plusの本体部が常駐し，バンク1はユーザのプログラムが置かれるTPA（トランジェント・プログラム・エリア）です（図15・8，図15・9）．

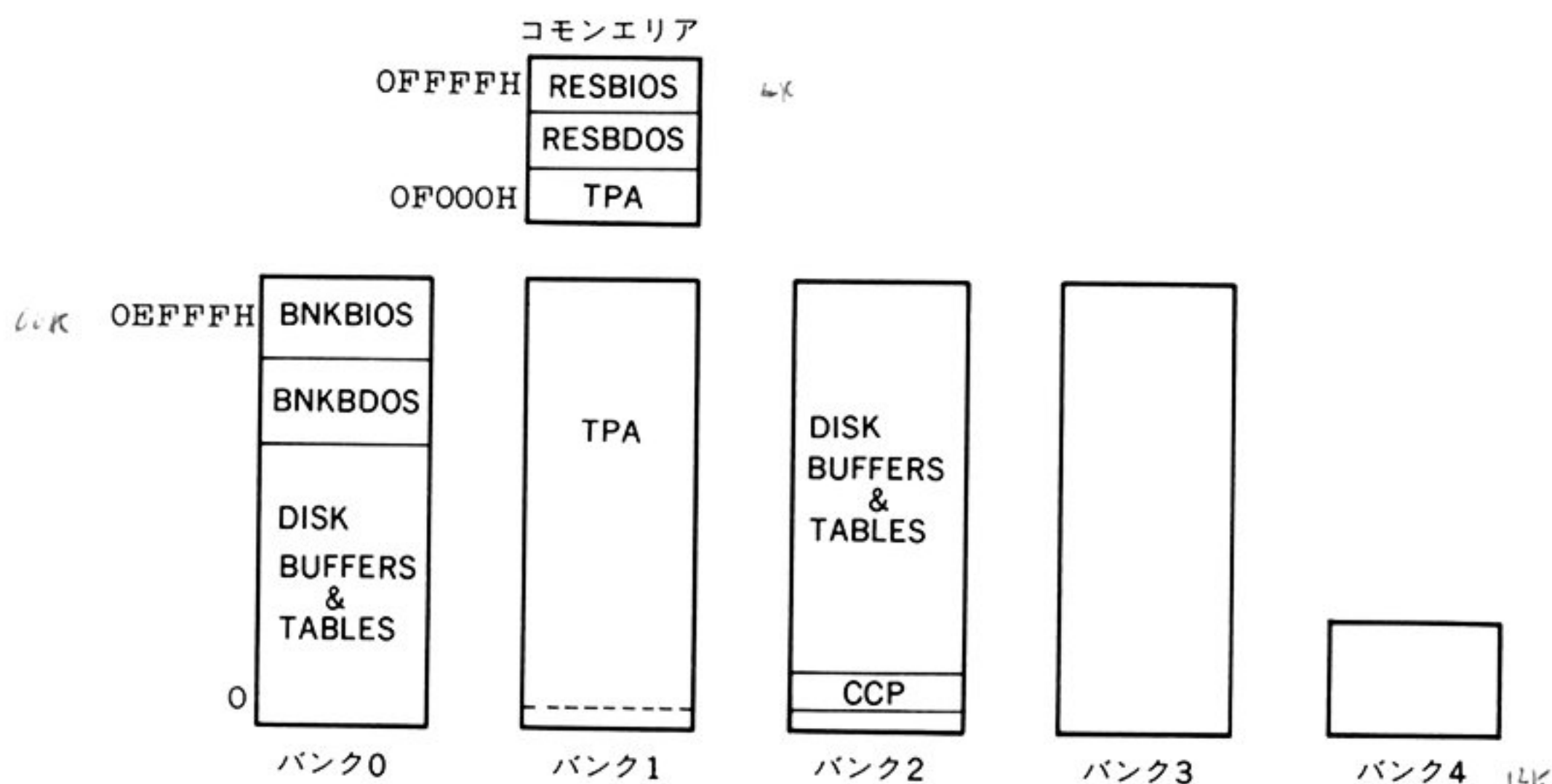



図 15・8 CP/M-Plusのバンク方式

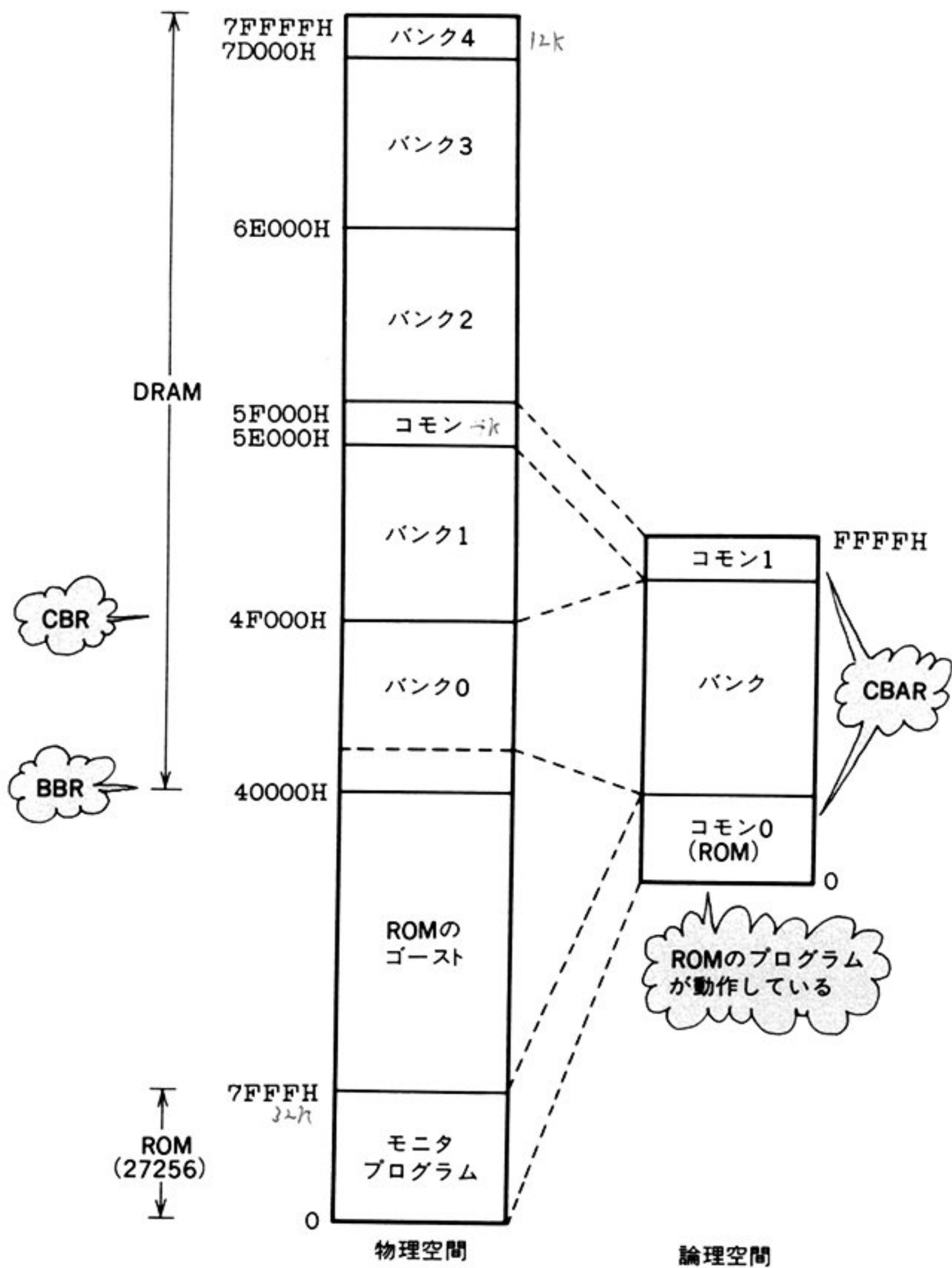



図 15・9　物理メモリマップと論理バンクの対応

一方，64180は，64Kバイトの論理アドレスをMMUにより1Mバイトの物理アドレスに拡張しています．1Mバイトの空間は，連続的で64Kバイトごとに区切られていません．この1Mバイトの空間を適切なところで分割し，CP/M-Plusに対応したバンクメモリにします．これは，64180のMMUを用いれば簡単に分割できますが，64180の物理メモリ空間とCP/M-Plusの論理メモリ空間をどのように対応させるかは，プログラムの実行効率のうえで重要です．その構成によっては次節の問題がおこることもあります．

カードパソコンでは，64180のMMUにより64Kバイトのメモリ空間を，4Kバイトのコモンエリアと60Kバイトのバンクエリアに分割しています．また，カードパソコンはHM50256を8個用いており，256Kバイトの物理メモリ空間があります．これをコモンエリア1つとバンクエリア5つに分割します．バンクエリアの1つは12Kバイトとなります．

分割はBIOSソフトウェア上での定義だけで物理空間と論理空間の対応が決まるので，ハードウェアでの対応が必要なわけではありません．

15・7　DMA 転送とバンク

CP/M-Plus はディスク・バッファによりディスク・アクセスのスループットを向上させていますが，この機能ではメモリ-FDC 間の DMA 転送およびメモリ↔メモリ間の DMA 転送が多用されます．このように DMA 転送は便利な機能ですが，前節の論理・物理のメモリ空間の対応でも考慮すべき点があります．

前節で述べたように，CP/M-Plus は 64180 の物理メモリ空間を分割したバンク方式による論理メモリ空間です．CP/M-Plus の本体である BDOS や TPA 内のユーザプログラムは，この論理的なメモリ空間の上で動作します．しかし，DMAC は MMU を通して物理アドレスを作っていません．直接，1 M バイトの物理アドレス空間に対して DMA 転送を実行します（**図 15・10**）．

前節のカードパソコンでのメモリ分割例で，コモンエリアとバンク 0 が物理空間上で連続した配置とした場合，CP/M-Plus はバンク 1 に TPA を置きますが，コモンエリアにも TPA の続きが置かれます．この TPA 上でアプリケーションのプログラムが動作しているとき，プログラムは TPA の端までディスクからデータをロードします．これは DMA 転送により実行されますが，データが多ければ E000H から F1FFH まで一度に転送します．このときの DMAC へ設定する物理アドレスは，バンク 1 に対応した値です．転送の最初の 4 K バイト（E000H～EFFFH）は正常に行われますが，F000H から F1FFH へ転送するはずのデータはコモンエリアではなく，バンク 2 に転送されバンク 2 の内容を破壊します．DMAC は，プログラム上で定義しているメモリの分割を考慮して転送しません．物理空間に対して転送します．

カードパソコンでは以上の問題により前項のメモリ配置とし，バンクエリア 1 とコモンエリアを論理空間だけでなく物理空間上でも連続した配置にしました．

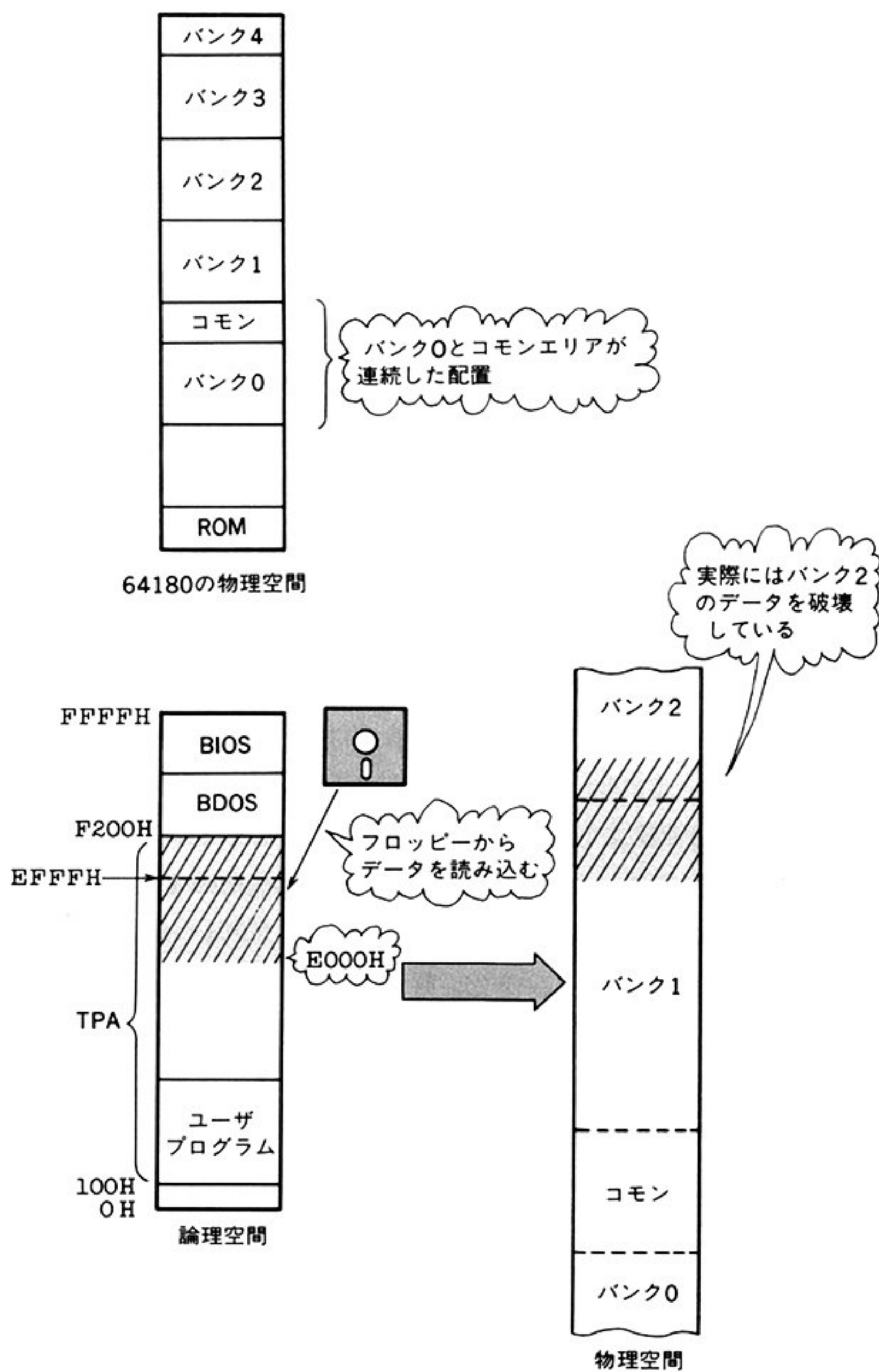



図 15・10　DMA 転送の問題がおこる例

15・8 CP/M-Plus起動時のメモリマップの変化

CP/M-Plusのバンクメモリの具体的な動作例として，CP/M-Plus起動時のメモリマップの変化を説明します．CP/M-PlusはDRAM上にロードされ動作しますが，64180のリセットからCP/M-Plusが起動するまでの動作は少々複雑です．

① 64180がリセットされると，MMUなどが初期化され，64180からはROMだけがアクセスできます．ROMにはIPL（イニシャルプログラムローダ）が書き込まれています．

② ROMのプログラムが動作し，MMUのレジスタを設定します．

③ フロッピーディスクの0トラック1セクタからコールドスタートローダを読み込み，コールドスタートローダにジャンプします．

④ コールドスタートローダは，最初にMMUのCBARを設定し直しROMを論理空間からはずします．

⑤ 次に，フロッピーディスクのトラック0セクタ2～セクタ16に入っているCP/Mローダを読み込み，CP/Mローダにジャンプします．

⑥ CP/MローダはフロッピーディスクのCPM3.SYSファイルを捜し，ファイルのヘッダレコード中の配置情報により，BNKBDOS，BNKBIOS，RESBDOS，RESBIOSをそれぞれ配置しながら読み込みます．その後，BIOSのBOOTコールにジャンプします．

⑦ BIOSのBOOTコールは周辺I/Oのイニシャライズ後，バンク1に切り替えます．

以上でCP/M-Plusの動作準備が完了します．この後，CCPコマンドを実行し"A>"のプロンプトが表示され，コマンド入力待ちになります（**図15・11**）．

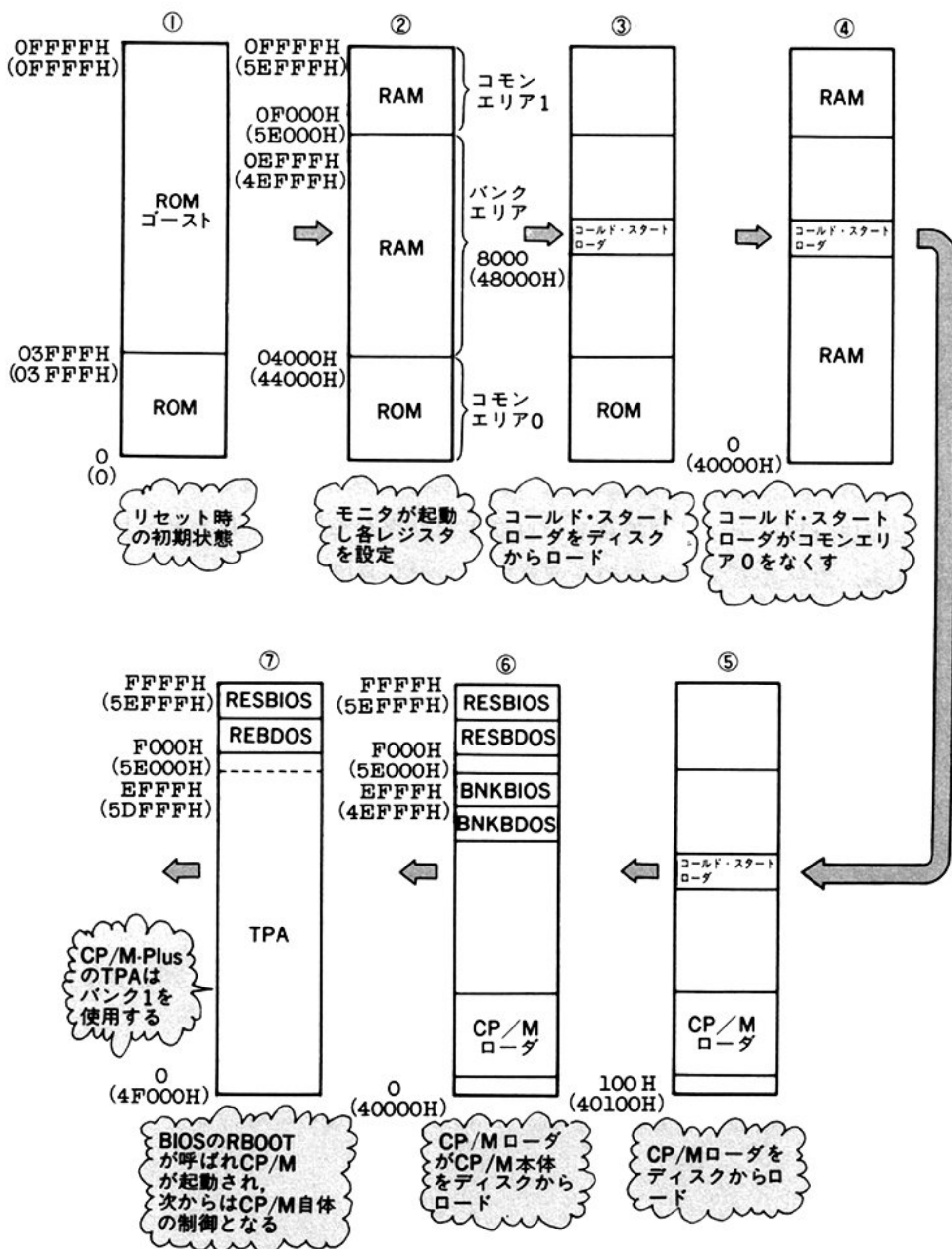



図 15・11　CP/M-Plus 起動時の論理メモリ内容の変化－(　)は物理アドレス－

15・9　64180内蔵I/OとBIOS

BIOSの中で64180の内蔵I/Oはどのようにプログラミングされているか，MMUとDMAC，ASCIについて取り上げます．

〔1〕 **MMU**　　バンク切替えのBIOSコール，SETBNKで設定します．各バンクに対応したBBRの値はまとめてテーブルにしておき，バンク番号からBBR値を変換できるようにします．このテーブルはDMACを設定するルーチンでも用います．CP/M-Plusシステムのメモリ構成を決める重要なルーチンですが，内容は図のとおり簡単です．

〔2〕 **DMAC**　　DMACはフロッピーディスクのBIOSコール，READ/WRITEやメモリ↔メモリ間のデータ転送のBIOSコール・MOVEで用います．ここではDMACの設定時に共通な論理－物理アドレス変換の部分だけを取り上げます．

BIOSがコールされたときに指定された転送アドレスはバンク番号と論理アドレスです．しかし，DMACは物理アドレスで転送するので，バンク番号と論理アドレスを計算して物理アドレスにします．この計算はMMUの設定で用いたテーブルからバンク番号に対応した値を取り出し，アドレスの上位と加算します．計算結果は物理アドレスとなりますが，DMA転送はコモンエリアに対して行われることもあるので，このときにはバンク番号を1にした計算をします．

ソースアドレス，ディスティネーションアドレスともアドレスを設定した後，DMACをスタートさせます．メモリ↔メモリ間転送の場合，転送はバーストモードで行われ，プログラムで転送終了を待つ必要はありません．

〔3〕 **ASCI**　　ASCIは文字入出力を行うBIOSコール，CONIN，CONOUTなどで用います．プログラム例はASCI1からの入力ルーチンです．プログラムは，文字が入力されたかをチェックするルーチンと，入力された文字をレジスタから取り出すルーチンに分かれます．このように入出力プログラムでも，内蔵I/Oでは短いプログラムで実現できます（図15・12）．

① バンク切替えプログラム

```
?bank:  PUSH DE
        PUSH HL
        LD   D,0
        LD   E,A
        LD   HL,bank table  ┐
        ADD  HL,DE          │ バンク番号に対応したテーブル値をロード
        LD   A,(HL)         ┘
        OUT0 (bbr),A        ←BBRにセットしバンク切替え
        POP  HL
        POP  DE
        RET
banktable: DB 40H,4FH,5FH,6EH,7DH
```

② DMA論理-物理アドレス変換プログラム

```
        LD   HL,(dmaaddress)
        PUSH HL
        LD   A,(bank)
        LD   C,A
        LD   A,0FFH         ┐
        CP   H              │
        JR   Z,cmn          │ DMAアドレスがコモンエリア内かをチェック
        JR   NC,n_cmn       ┘
cmn:    LD   C,1            ←コモンエリア内であればバンク番号を1にする
n_cmn:  LD B,0
        LD   HL,banktable   ┐
        ADD  HL,BC          │ バンク番号に対応したテーブル値をロード
        LD   C,(HL)         ┘
        LD   B,16           ┐
        MLT  BC             ┘ テーブル値×16
        POP  HL
        LD   A,H            ┐
        ADD  A,C            │
        LD   H,A            │ テーブル値とDMAアドレスを加算
        LD   A,B            │
        ADC  A,0            ┘
```

③ ASCI入力フラグテストプログラム

```
asci_st: IN0 A,(stat1)
         AND 80H            ←RDRFフラグをテスト
         RET Z              ←フラグが0ならAを0にしてリターン
         OR  0FFH
         RET                ←フラグが1ならAをFFHにしてリターン
```

④ ASCI入力プログラム

```
asci_in: CALL asci_st       ┐
         JR   Z,asci_in     ┘ 文字を受信するまで待つ
         IN0  A,(tsr1)      ←文字をロード
         RET
```

図 15・12 内蔵I/Oのプログラム

付　録

1 命令一覧

命令セットの一覧表の中で使用される記号を以下に説明します．

1. レジスタ指定

g，g′，ww，xx，yy，zz，はレジスタを指定する記号です．g，g′は8ビットのレジスタ，ww，xx，yy，zz，は16ビットのレジスタペアを指定します．おのおの対応するレジスタは下記のとおりです．

g,g′	Reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

ww	Reg.
00	BC
01	DE
10	HL
11	SP

xx	Reg.
00	BC
01	DE
10	IX
11	SP

yy	Reg.
00	BC
01	DE
10	IY
11	SP

zz	Reg.
00	BC
01	DE
10	HL
11	AF

注） ww，xx，yy，zz に H または L を付加することにより，16 ビットレジスタの上位8ビットまたは下位8ビットを表します（例．wwH，IXL）．

2. ビット指定

b は，ビット操作命令におけるビットオペランドが何ビット目かを指定する記号です．おのおの対応するビットは右表のとおりです．

b	Bit
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

3. コンディション指定

f は，演算の結果を判定して命令を実行する場合の条件を指定する記号です．おのおの対応するコンディションは右表のとおりです．

f	Condition
000	NZ　non zero
001	Z　zero
010	NC　non carry
011	C　carry
100	PO　parity odd
101	PE　parity even
110	P　sign plus
111	M　sign minus

4．リスタートアドレス

v は，リスタート命令のリスタートアドレスを指定する記号です．おのおの対応するアドレスは右表のとおりです．

v	Address
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

5. フ ラ グ

フラグの変化を示す記号について説明します．

・：その命令によってフラグは変化しない
×：その命令によってフラグの変化は不定
↕：その命令によって演算結果に従いフラグは変化
S：その命令によって“1”にセット
R：その命令によって“0”にリセット
P：その命令によってパリティフラグとして変化
V：その命令によってオーバフローフラグとして変化

6. そ の 他

(　　)$_M$：(　　) 内の内容をアドレスとするメモリを表します．

(　　)$_I$：(　　) 内の内容をアドレスとするI/Oを表します

m or n：8ビットの値

m n：16ビットの値

r：rの添字がついていると，8ビットレジスタを表します

R：Rの添字がついていると，16ビットレジスタを表します

b・(　　)$_M$：(　　) 内の内容をアドレスとするメモリのbで指定されるビットを表します

b：gr：grで指定されるレジスタの内容のbで指定されるビットを表します

d or j：符号つき8ビットディスプレースメント

S：ソースアドレシングモード

D：ディスティネーションアドレシングモード

・：AND演算

＋：OR演算

⊕：EXCLUSIVE OR演算

(1) 演　算　命　令

1. 算術論理演算（8-bit）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ 7 S	6 Z	4 H	2 P/V	1 N	0 C
ADD	ADD A, g	10 000 g	1	4	Ar+gr→Ar	↕	↕	↕	V	R	↕
	ADD A, (HL)	10 000 110	1	6	Ar+(HL)$_M$→Ar	↕	↕	↕	V	R	↕
	ADD A, m	11 000 110 〈m〉	2	6	Ar+m→Ar	↕	↕	↕	V	R	↕
	ADD A, (IX+d)	11 011 101 10 000 110 〈d〉	3	14	Ar+(IX+d)$_M$→Ar	↕	↕	↕	V	R	↕
	ADD A, (IY+d)	11 111 101 10 000 110 〈d〉	3	14	Ar+(IY+d)$_M$→Ar	↕	↕	↕	V	R	↕
ADC	ADC A, g	10 001 g	1	4	Ar+gr+c→Ar	↕	↕	↕	V	R	↕
	ADC A, (HL)	10 001 110	1	6	Ar+(HL)$_M$+c→Ar	↕	↕	↕	V	R	↕
	ADC A, m	11 001 110 〈m〉	2	6	Ar+m+c→Ar	↕	↕	↕	V	R	↕
	ADC A, (IX+d)	11 011 101 10 001 110 〈d〉	3	14	Ar+(IX+d)$_M$+c→Ar	↕	↕	↕	V	R	↕
	ADC A, (IY+d)	11 111 101 10 001 110 〈d〉	3	14	Ar+(IY+d)$_M$+c→Ar	↕	↕	↕	V	R	↕

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
AND	AND g	10 100 g	1	4	Ar·gr→Ar	↕	↕	S	P	R	R
	AND (HL)	10 100 110	1	6	Ar·$(HL)_M$→Ar	↕	↕	S	P	R	R
	AND m	11 100 110 〈m〉	2	6	Ar·m→Ar	↕	↕	S	P	R	R
	AND(IX+d)	11 011 101 10 100 110 〈d〉	3	14	Ar·$(IX+d)_M$→Ar	↕	↕	S	P	R	R
	AND (IY+d)	11 111 101 10 100 110 〈d〉	3	14	Ar·$(IY+d)_M$→Ar	↕	↕	S	P	R	R
Compare	CP g	10 111 g	1	4	Ar−gr	↕	↕	↕	V	S	↕
	CP (HL)	10 111 110	1	6	Ar−$(HL)_M$	↕	↕	↕	V	S	↕
	CP m	11 111 110 〈m〉	2	6	Ar−m	↕	↕	↕	V	S	↕
	CP (IX+d)	11 011 101 10 111 110 〈d〉	3	14	Ar−$(IX+d)_M$	↕	↕	↕	V	S	↕
	CP (IY+d)	11 111 101 10 111 110 〈d〉	3	14	Ar−$(IY+d)_M$	↕	↕	↕	V	S	↕
COMPLE- MENT	CPL	00 101 111	1	3	$\overline{Ar}$→Ar	·	·	S	·	S	·
DEC	DEC g	00 g 101	1	4	gr−1→gr	↕	↕	↕	V	S	·
	DEC (HL)	00 110 101	1	10	$(HL)_M$−1→$(HL)_M$	↕	↕	↕	V	S	·
	DEC (IX+d)	11 011 101 00 110 101 〈d〉	3	18	$(IX+d)_M$−1→ $(IX+d)_M$	↕	↕	↕	V	S	·
	DEC (IY+d)	11 111 101 00 110 101 〈d〉	3	18	$(IY+d)_M$−1→ $(IY+d)_M$	↕	↕	↕	V	S	·
INC	INC g	00 g 100	1	4	gr+1→gr	↕	↕	↕	V	R	·
	INC (HL)	00 110 100	1	10	$(HL)_M$+1→$(HL)_M$	↕	↕	↕	V	R	·
	INC (IX+d)	11 011 101 00 110 100 〈d〉	3	18	$(IX+d)_M$+1→ $(IX+d)_M$	↕	↕	↕	V	R	·
	INC (IY+d)	11 111 101 00 110 100 〈d〉	3	18	$(IY+d)_M$ +1→ $(IY+d)_M$	↕	↕	↕	V	R	·
MULT	MLT ww	11 101 101 01 ww1 100	2	17	wwHr×wwLr→ww_R	·	·	·	·	·	·
NEGATE	NEG	11 101 101 01 000 100	2	6	0−Ar→Ar	↕	↕	↕	V	S	↕

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
OR	OR g	10 110 g	1	4	Ar+gr→Ar	↕	↕	R	P	R	R
	OR (HL)	10 110 110	1	6	Ar+(HL)$_M$→Ar	↕	↕	R	P	R	R
	OR m	11 110 110 〈m〉	2	6	Ar+m→Ar	↕	↕	R	P	R	R
	OR (IX+d)	11 011 101 10 110 110 〈d〉	3	14	Ar+(IX+d)$_M$→Ar	↕	↕	R	P	R	R
	OR (IY+d)	11 111 101 10 110 110 〈d〉	3	14	Ar+(IY+d)$_M$→Ar	↕	↕	R	P	R	R
SUB	SUB g	10 010 g	1	4	Ar−gr→Ar	↕	↕	↕	V	S	↕
	SUB (HL)	10 010 110	1	6	Ar−(HL)$_M$→Ar	↕	↕	↕	V	S	↕
	SUB m	11 010 110 〈m〉	2	6	Ar−m→Ar	↕	↕	↕	V	S	↕
	SUB (IX+d)	11 011 101 10 010 110 〈d〉	3	14	Ar−(IX+d)$_M$→→Ar	↕	↕	↕	V	S	↕
	SUB (IY+d)	11 111 101 10 010 110 〈d〉	3	14	Ar−(IY+d)$_M$→Ar	↕	↕	↕	V	S	↕
SUBC	SBC A,g	10 011 g	1	4	Ar−gr−c→Ar	↕	↕	↕	V	S	↕
	SBC A,(HL)	10 011 110	1	6	Ar−(HL)$_M$−c→Ar	↕	↕	↕	V	S	↕
	SBC A,m	11 011 110 〈m〉	2	6	Ar−m−c→Ar	↕	↕	↕	V	S	↕
	SBC A,(IX+d)	11 011 101 10 011 110 〈d〉	3	14	Ar−(IX+d)$_M$−c→Ar	↕	↕	↕	V	S	↕
	SBC A,(IY+d)	11 111 101 10 011 110 〈d〉	3	14	Ar−(IY+d)$_M$−c→Ar	↕	↕	↕	V	S	↕
TEST	TST g	11 101 101 00 g 100	2	7	Ar・gr	↕	↕	S	P	R	R
	TST (HL)	11 101 101 00 110 100	2	10	Ar・(HL)$_M$	↕	↕	S	P	R	R
	TST m	11 101 101 01 100 100 〈m〉	3	9	Ar・m	↕	↕	S	P	R	R
XOR	XOR g	10 101 g	1	4	Ar⊕gr→Ar	↕	↕	R	P	R	R
	XOR (HL)	10 101 110	1	6	Ar⊕(HL)$_M$→Ar	↕	↕	R	P	R	R
	XOR m	11 101 110 〈m〉	2	6	Ar⊕m→Ar	↕	↕	R	P	R	R
	XOR (IX+d)	11 011 101 10 101 110	3	14	Ar⊕(IY+d)$_M$→Ar	↕	↕	R	P	R	R

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
	XOR(IY+d)	〈d〉 11 111 101 10 101 110 〈d〉	3	14	Ar⊕(IY+d)M→Ar	↕	↕	R	P	R	R

2. ローテートおよびシフト命令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Rotate and Shift Data	RLA	00 010 111	1	3	C b7 ← b0	・	・	R	・	R	↕
	RL g	11 001 011 00 010 g	2	7		↕	↕	R	P	R	↕
	RL (HL)	11 001 011 00 010 110	2	13		↕	↕	R	P	R	↕
	RL (IX+d)	11 011 101 11 001 011 〈d〉 00 010 110	4	19		↕	↕	R	P	R	↕
	RL (IY+d)	11 111 101 11 001 011 〈d〉 00 010 110	4	19		↕	↕	R	P	R	↕
	RLCA	00 000 111	1	3	C b7 ← b0	・	・	R	・	R	↕
	RLC g	11 001 011 00 000 g	2	7		↕	↕	R	P	R	↕
	RLC (HL)	11 001 011 00 000 110	2	13		↕	↕	R	P	R	↕
	RLC (IX+d)	11 011 101 11 001 011 〈d〉 00 000 110	4	19		↕	↕	R	P	R	↕
	RLC (IY+d)	11 111 101 11 001 011 〈d〉 00 000 110	4	19	Ar b7 b0 (HL)M b7 b0	↕	↕	R	P	R	↕
	RLD	11 101 101 01 101 111	2	16		↕	↕	R	P	R	・
	RRA	00 011 111	1	3	b7 → b0 C	・	・	R	・	R	↕
	RR g	11 001 011 00 011 g	2	7		↕	↕	R	P	R	↕
	RR (HL)	11 001 011 00 011 110	2	13		↕	↕	R	P	R	↕

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
	RR (IX+d)	11 011 101 11 001 011 〈d〉 00 011 110	4	19		↕	↕	R	P	R	↕
	RR (IY+d)	11 111 101 11 001 011 〈d〉 00 011 110	4	19		↕	↕	R	P	R	↕
	RRCA	00 001 111	1	3	b7 → b0 C	·	·	R	·	R	↕
	RRC g	11 001 011 00 001 g	2	7		↕	↕	R	P	R	↕
	RRC (HL)	11 001 011 00 001 110	2	13		↕	↕	R	P	R	↕
	RRC (IX+d)	11 011 101 11 001 011 〈d〉 00 001 110	4	19		↕	↕	R	P	R	↕
	RRC (IY+d)	11 111 101 11 001 011 〈d〉 00 001 110	4	19		↕	↕	R	P	R	↕
Rotate and Shift Data	RRD	11 101 101 01 100 111	2	16	Ar b7 b0 (HL)M b7 b0	↕	↕	R	P	R	·
	SLA g	11 001 011 00 100 g	2	7	C b7 b0 ← 0	↕	↕	R	P	R	↕
	SLA (HL)	11 001 011 00 100 110	2	13		↕	↕	R	P	R	↕
	SLA (IX+d)	11 011 101 11 001 011 〈d〉 00 100 110	4	19		↕	↕	R	P	R	↕
	SLA (IY+d)	11 111 101 11 001 011 〈d〉 00 100 110	4	19		↕	↕	R	P	R	↕
	SRA g	11 001 011 00 101 g	2	7	b7 b0 C	↕	↕	R	P	R	↕
	SRA (HL)	11 001 011 00 101 110	2	13		↕	↕	R	P	R	↕
	SRA (IX+d)	11 011 101 11 001 011 〈d〉 00 101 110	4	19		↕	↕	R	P	R	↕
	SRA (IY+d)	11 111 101 11 001 011	4	19		↕	↕	R	P	R	↕

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
		〈d〉 00 101 110									
	SRL g	11 001 011 00 111 g	2	7	0→[b7 … b0]→C	↕	↕	R	P	R	↕
	SRL (HL)	11 001 011 00 111 110	2	3		↕	↕	R	P	R	↕
	SRL (IX+d)	11 011 101 11 001 011 〈d〉 00 111 110	4	19		↕	↕	R	P	R	↕
	SRL (IY+d)	11 111 101 11 001 011 〈d〉 00 111 110	4	19		↕	↕	R	P	R	↕

3. ビット操作命令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Bit Set	SET b, g	11 001 011 11 b g	2	7	1→b·gr	·	·	·	·	·	·
	SET b, (HL)	11 001 011 11 b 110	2	13	1→b·$(HL)_M$	·	·	·	·	·	·
	SET b, (IX+d)	11 011 101 11 001 011 〈d〉 11 b 110	4	19	1→b·$(IX+d)_M$	·	·	·	·	·	·
	SET b, (IY+d)	11 111 101 11 001 011 〈d〉 11 b 110	4	19	1→b·$(IY+d)_M$	·	·	·	·	·	·
Bit Reset	RES b, g	11 001 011 10 b g	2	7	0→b·gr	·	·	·	·	·	·
	RES b, (HL)	11 001 011 10 b 110	2	13	0→b·$(HL)_M$	·	·	·	·	·	·
	RES b, (IX+d)	11 011 101 11 001 011 〈d〉 10 b 110	4	19	0→b·$(IX+d)_M$	·	·	·	·	·	·
	RES b, (IY+d)	11 111 101	4	19	0→b·$(IY+d)_M$	·	·	·	·	·	·

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
		11 001 011 〈d〉 10 b 110									
Bit Test	BIT b, g	11 001 011 01 b g	2	6	$\overline{b \cdot gr} \rightarrow z$	X	↕	S	X	R	·
	BIT b, (HL)	11 001 011 01 b 110	2	9	$\overline{b \cdot (HL)_M} \rightarrow z$	X	↕	S	X	R	·
	BIT b, (IX+d)	11 011 101 11 001 011 〈d〉 01 b 110	4	15	$\overline{b \cdot (IX+d)_M} \rightarrow z$	X	↕	S	X	R	·
	BIT b, (IY+d)	11 111 101 11 001 011 〈d〉 01 b 110	4	15	$\overline{b \cdot (IY+d)_M} \rightarrow z$	X	↕	S	X	R	·

4. 演 算 命 令 (16-bit)

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
ADD	ADD HL, ww	00 ww1 001	1	7	$HL_R + ww_R \rightarrow HL_R$	·	·	X	·	R	↕
	ADD IX, xx	11 011 101 00 xx1 001	2	10	$IX_R + xx_R \rightarrow IX_R$	·	·	X	·	R	↕
	ADD IY, yy	11 111 101 00 yy1 001	2	10	$IY_R + yy_R \rightarrow IY_R$	·	·	X	·	R	↕
ADC	ADC HL, ww	11 101 101 01 ww1 010	2	10	$HL_R + ww_R + c \rightarrow HL_R$	↕	↕	X	V	R	↕
DEC	DEC ww	00 ww1 011	1	4	$ww_R - 1 \rightarrow ww_R$	·	·	·	·	·	·
	DEC IX	11 011 101 00 101 011	2	7	$IX_R - 1 \rightarrow IX_R$	·	·	·	·	·	·
	DEC IY	11 111 101 00 101 011	2	7	$IY_R - 1 \rightarrow IY_R$	·	·	·	·	·	·
INC	INC ww	00 ww0 011	1	4	$ww_R + 1 \rightarrow ww_R$	·	·	·	·	·	·
	INC IX	11 011 101 00 100 011	2	7	$IX_R + 1 \rightarrow IX_R$	·	·	·	·	·	·
	INC IY	11 111 101 00 100 011	2	7	$IY_R + 1 \rightarrow IY_R$	·	·	·	·	·	·
SBC	SBC HL, ww	11 101 101 01 ww0 010	2	10	$HL_R - ww_R - c \rightarrow HL_R$	↕	↕	X	V	S	↕

（2） 転 送 命 令

1. 8ビット転送

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Load 8-bit Data	LD A, I	11 101 101 01 010 111	2	6	Ir→Ar ①	↕	↕	R	IEF_2	R	·
	LD A, R	11 101 101 01 011 111	2	6	Rr→Ar ①	↕	↕	R	IEF_2	R	·
	LD A, (BC)	00 001 010	1	6	$(BC)_M$→Ar	·	·	·	·	·	·
	LD A, (DE)	00 011 010	1	6	$(DE)_M$→Ar	·	·	·	·	·	·
	LD A, (mn)	00 111 010 〈n〉 〈m〉	3	12	$(mn)_M$→Ar	·	·	·	·	·	·
	LD I, A	11 101 101 01 000 111	2	6	Ar→Ir	·	·	·	·	·	·
	LD R, A	11 101 101 01 001 111	2	6	Ar→Rr	·	·	·	·	·	·
	LD(BC), A	00 000 010	1	7	Ar→$(BC)_M$	·	·	·	·	·	·
	LD(DE), A	00 010 110	1	7	Ar→$(DE)_M$	·	·	·	·	·	·
	LD(mn), A	00 110 010 〈n〉 〈m〉	1	13	Ar→$(mn)_M$	·	·	·	·	·	·
	LD g, g′	01 g g′	1	4	gr′→gr	·	·	·	·	·	·
	LD g, (HL)	01 g 110	1	6	$(HL)_M$→gr	·	·	·	·	·	·
	LD g, m	00 g 110 〈m〉	2	6	m→gr	·	·	·	·	·	·
	LD g, (IX+d)	11 011 101 01 g 110 〈d〉	3	14	$(IX+d)_M$→gr	·	·	·	·	·	·
	LD g, (IY+d)	11 111 101 01 g 110 〈d〉	3	14	$(IY+d)_M$→gr	·	·	·	·	·	·
	LD (HL), m	00 110 110 〈m〉	2	9	m→$(HL)_M$	·	·	·	·	·	·
	LD(IX+d), m	11 011 101 00 110 110 〈d〉 〈m〉	4	15	m→$(IX+d)_M$	·	·	·	·	·	·
	LD(IY+d), m	11 111 101 00 110 110 〈d〉 〈m〉	4	15	m→$(IY+d)_M$	·	·	·	·	·	·
	LD(HL), g	01 110 g	1	7	gr→$(HL)_M$	·	·	·	·	·	·
	LD(IX+d), g	11 011 101 01 110 g 〈d〉	3	15	gr→$(IX+d)_M$	·	·	·	·	·	·
	LD(IY+d), g	11 111 101 01 110 g 〈d〉	3	15	gr→$(IY+d)_M$	·	·	·	·	·	·

① LD A, I またはLD A, Rの命令の最後では，割込みはサンプルされません．

2. 16ビット転送

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Load 16-bit Data	LD ww,mn	00 ww0 001 〈n〉 〈m〉	3	9	mn→ww_R	·	·	·	·	·	·
	LD IX,mn	11 011 101 00 100 001 〈n〉 〈m〉	4	12	mn→IX_R	·	·	·	·	·	·
	LD IY,mn	11 111 101 00 100 001 〈n〉 〈m〉	4	12	mn→IY_R	·	·	·	·	·	·
	LD SP,HL	11 111 001	1	4	HL_R→SP_R	·	·	·	·	·	·
	LD SP,IX	11 011 101 11 111 001	2	7	IX_R→SP_R	·	·	·	·	·	·
	LD SP,IY	11 111 101 11 111 001	2	7	IY_R→SP_R	·	·	·	·	·	·
	LDww,(mn)	11 101 101 01 ww1 011 〈n〉 〈m〉	4	18	$(mn+1)_M$→wwHr $(mn)_M$→wwLr	·	·	·	·	·	·
	LD HL,(mn)	00 101 010 〈n〉 〈m〉	3	15	$(mn+1)_M$→Hr $(mn)_M$→Lr	·	·	·	·	·	·
	LD IX,(mn)	11 011 101 00 101 010 〈n〉 〈m〉	4	18	$(mn+1)_M$→IXHr $(mn)_M$→IXLr	·	·	·	·	·	·
	LD IY,(mn)	11 111 101 00 101 010 〈n〉 〈m〉	4	18	$(mn+1)_M$→IYHr $(mn)_M$→IYLr	·	·	·	·	·	·
	LD(mn),ww	11 101 101 01 ww0 011 〈n〉 〈m〉	4	19	wwHr→$(mn+1)_M$ wwLr→$(mn)_M$	·	·	·	·	·	·
	LD(mn),HL	00 100 010 〈n〉 〈m〉	3	16	Hr→$(mn+1)_M$ Lr→$(mn)_M$	·	·	·	·	·	·
	LD(mn),IX	11 011 101 00 100 010 〈n〉 〈m〉	4	19	IXHr→$(mn+1)_M$ IXLr→$(mn)_M$	·	·	·	·	·	·
	LD(mn),IY	11 111 101 00 100 010 〈n〉 〈m〉	4	19	IYHr→$(mn+1)_M$ IYLr→$(mn)_M$	·	·	·	·	·	·

3. ブロック転送

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Block Transfer Search Data	CPD	11 101 101 10 101 001	2	12	$Ar-(HL)_M$ $BC_R-1\to BC_R$ $HL_R-1\to HL_R$	↕	③ ↕	↕	② ↕	S	·
	CPDR	11 101 101 10 111 001	2	14 12	$BC_R\neq0$ $Ar\neq(HL)_M$ $BC_R=0$ or $Ar=(HL)_M$ Q ($Ar-(HL)_M$ $BC_R-1\to BC_R$ $HL_R-1\to HL_R$) Repeat Q until $Ar=(HL)_M$ or $BC_R=0$	↕	③ ↕	↕	② ↕	S	·
	CPI	11 101 101 10 100 001	2	12	$Ar-(HL)_M$ $BC_R-1\to BC_R$ $HL_R+1\to HL_R$	↕	③ ↕	↕	② ↕	S	·
	CPIR	11 101 101 10 110 001	2	14 12	$BC_R\neq0$ $Ar\neq(HL)_M$ $BC_R=0$ or $Ar=(HL)_M$ Q ($Ar-(HL)_M$ $BC_R-1\to BC_R$ $HL_R+1\to HL_R$) Repeat Q until $Ar=(HL)_M$ or $BC_R=0$	↕	③ ↕	↕	② ↕	S	·
	LDD	11 101 101 10 101 000	2	12	$(HL)_M\to(DE)_M$ $BC_R-1\to BC_R$ $DE_R-1\to DE_R$ $HL_R-1\to HL_R$	·	·	R	② ↕	R	·
	LDDR	11 101 101 10 111 000	2	14($BC_R\neq0$) 12($BC_R=0$)	Q ($(HL)_M\to(DE)_M$ $BC_R-1\to BC_R$ $DE_R-1\to DE_R$ $HL_R-1\to HL_R$) Repeat Q until $BC_R=0$	·	·	R	R	R	·
	LDI	11 101 101 10 100 000	2	12	$(HL)_M\to(DE)_M$ $BC_R-1\to BC_R$ $DE_R+1\to DE_R$ $HL_R+1\to HL_R$	·	·	R	② ↕	R	·
	LDIR	11 101 101 10 110 000	2	14($BC_R\neq0$) 12($BC_R=0$)	Q ($(HL)_M\to(DE)_M$ $BC_R-1\to BC_R$ $DE_R+1\to DE_R$ $HL_R+1\to HL_R$) Repeat Q until $BC_R=0$	·	·	R	R	R	·

② $P/V=0 : BC_R-1=0$　　③ $Z=1 : Ar=(HL)_M$
　$P/V=1 : BC_R-1\neq0$　　　$Z=0 : Ar\neq(HL)_M$

4. スタック処理および交換

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ 7 S	6 Z	4 H	2 P/V	1 N	0 C
PUSH	PUSH zz	11 zz0 101	1	11	zzLr→$(SP-2)_M$ zzHr→$(SP-1)_M$ $SP_R-2→SP_R$	·	·	·	·	·	·
	PUSH IX	11 011 101 11 100 101	2	14	IXLr→$(SP-2)_M$ IXHr→$(SP-1)_M$ $SP_R-2→SP_R$	·	·	·	·	·	·
	PUSH IY	11 111 101 11 100 101	2	14	IYLr→$(SP-2)_M$ IYHr→$(SP-1)_M$ $SP_R-2→SP_R$	·	·	·	·	·	·
POP	POP zz	11 zz0 001	1	9	$(SP+1)_M$→zzHr ④ $(SP)_M$→zzLr $SP_R+2→SP_R$	·	·	·	·	·	·
	POP IX	11 011 101 11 100 001	2	12	$(SP+1)_M$→IXHr $(SP)_M$→IXLr $SP_R+2→SP_R$	·	·	·	·	·	·
	POP IY	11 111 101 11 100 001	2	12	$(SP+1)_M$→IYHr $(SP)_M$→IYLr $SP_R+2→SP_R$	·	·	·	·	·	·
Exchange	EX AF, AF	00 001 000	1	4	$AF_R↔AF_R'$	·	·	·	·	·	·
	EX DE, HL	11 101 011	1	3	$DE_R↔HL_R$	·	·	·	·	·	·
	EXX	11 011 001	1	3	$BC_R↔BC_R'$ $DE_R↔DE_R'$ $HL_R↔HL_R'$	·	·	·	·	·	·
	EX (SP), HL	11 100 011	1	16	Hr↔$(SP+1)_M$ Lr↔$(SP)_M$	·	·	·	·	·	·
	EX (SP), IX	11 011 101 11 100 011	2	19	IXHr↔$(SP+1)_M$ IXLr↔$(SP)_M$	·	·	·	·	·	·
	EX (SP), IY	11 111 101 11 100 011	2	19	IYHr↔$(SP+1)_M$ IYLr↔$(SP)_M$	·	·	·	·	·	·

④ POP AF では，スタックの内容がフラグに書き込まれます．

(3) プログラム制御命令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ 7 S	6 Z	4 H	2 P/V	1 N	0 C
Call	CALL mn	11 001 101 〈n〉 〈m〉	3	16	PCHr→$(SP-1)_M$ PCLr→$(SP-2)_M$ mn→PC_R $SP_R-2→SP_R$	·	·	·	·	·	·
	CALL f, mn	11 f 100 〈n〉 〈m〉	3	6(f:false) 16(f:true)	continue:f is false CALL mn:f is true	·	·	·	·	·	·
Jump	DJNZ j	00 010 000 〈j-2〉	2 2	9(Br≠0) 7(Br=0)	Br−1→Br continue:Br=0 PC_R+j→PC_R:Br≠0	·	·	·	·	·	·
	JP f, mn	11 f 010 〈n〉 〈m〉	3 3	6(f:false) 9(f:true)	mn→PC_R:f is true continue:f is false	·	·	·	·	·	·
	JP mn	11 000 011 〈n〉 〈m〉	3	9	mn→PC_R	·	·	·	·	·	·
	JP (HL)	11 101 001	1	3	HL_R→PC_R	·	·	·	·	·	·
	JP (IX)	11 011 101 11 101 001	2	6	IX_R→PC_R	·	·	·	·	·	·
	JP (IY)	11 111 101 11 101 001	2	6	IY_R→PC_R	·	·	·	·	·	·
	JR j	00 011 000 〈j-2〉	2	8	PC_R+j→PC_R	·	·	·	·	·	·
	JR C, j	00 111 000 〈j-2〉	2 2	6 8	continue:C=0 PC_R+j→PC_R:C=1	·	·	·	·	·	·
	JR NC, j	00 110 000 〈j-2〉	2 2	6 8	continue:C=1 PC_R+j→PC_R:C=0	·	·	·	·	·	·
	JR Z, j	00 101 000 〈j-2〉	2 2	6 8	continue:Z=0 PC_R+j→PC_R:Z=1	·	·	·	·	·	·
	JR NZ, j	00 100 000 〈j-2〉	2 2	6 8	continue: Z=1 PC_R+j→PC_R:Z=0	·	·	·	·	·	·
Return	RET	11 001 001	1	9	$(SP)_M$→PCLr $(SP+1)_M$→PCHr $SP_R+2→SP_R$	·	·	·	·	·	·
	RET f	11 f 000	1 1	5(f:false) 10(f:true)	continue:f is false RET:f is true	·	·	·	·	·	·
	RETI	11 101 101 01 001 101	2	12(RI) 22(Z)	$(SP)_M$→PCLr $(SP+1)_M$→PCHr $SP_R+2→SP_R$	·	·	·	·	·	·
	RETN	11 101 101 01 000 101	2	12	$(SP)_M$→PCLr $(SP+1)_M$→PCHr $SP_R+2→SP_R$ $IEF_2→IEF_1$	·	·	·	·	·	·

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Restart	RST v	11 v 111	1	11	$PCHr \rightarrow (SP-1)_M$ $PCLr \rightarrow (SP-2)_M$ $0 \rightarrow PCHr$ $v \rightarrow PCLr$ $SP_R - 2 \rightarrow SP_R$	·	·	·	·	·	·

(4) I/O 命 令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
INPUT	IN A,(m)	11 011 011 〈m〉	2	9	$(Am)_I \rightarrow Ar$ $m \rightarrow A_0 \sim A_7$ $Ar \rightarrow A_8 \sim A_{15}$	·	·	·	·	·	·
	IN g,(C)	11 101 101 01 g 000	2	9	$(BC)_I \rightarrow gr$ g ＝1 1 0 : Only the flags will change. $Cr \rightarrow A_0 \sim A_7$ $Br \rightarrow A_8 \sim A_{15}$	↕	↕	R	P	R	·
	INO g,(m)	11 101 101 00 g 000 〈m〉	3	12	$(00m)_X \rightarrow gr$ g＝110:Only the flags will change. $m \rightarrow A_0 \sim A_7$ $00 \rightarrow A_8 \sim A_{15}$	↕	↕	R	P	R	·
	IND	11 101 101 10 101 010	2	12	$(BC)_I \rightarrow (HL)_M$ $HL_R - 1 \rightarrow HL_R$ $Br - 1 \rightarrow Br$ $Cr \rightarrow A_0 \sim A_7$ $Br \rightarrow A_8 \sim A_{15}$	X	⑤ ↕	X	X	⑥ ↕	X
	INDR	11 101 101 10 111 010	2	14(Br≠0) 12(Br＝0)	Q { $(BC)_I \rightarrow (HL)_M$ $HL_R - 1 \rightarrow HL_R$ $Br - 1 \rightarrow Br$ } Repeat Q until Br＝0 $Cr \rightarrow A_0 \sim A_7$ $Br \rightarrow A_8 \sim A_{15}$	X	S	X	X	⑥ ↕	X
	INI	11 101 101 10 100 010	2	12	$(BC)_I \rightarrow (HL)_M$ $HL_R + 1 \rightarrow HL_R$ $Br - 1 \rightarrow Br$	X	⑤ ↕	X	X	⑥ ↕	X

⑤ Z＝1 : Br－1＝0
　 Z＝0 : Br－1≠0

⑥ N＝1 : MSB of Data＝1
　 N＝0 : MSB of Data＝0

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
	INIR	11 101 101 10 110 010	2	14(Br≠0) 12(Br=0)	$Cr \to A_0 \sim A_7$ $Br \to A_8 \sim A_{15}$ Q($(BC)_I \to (HL)_M$ $HL_R + 1 \to HL_R$ $Br - 1 \to Br$) Repeat Q until Br=0 $Cr \to A_0 \sim A_7$ $Br \to A_8 \sim A_{15}$	X	S	X	X	⑥ ↕	X
OUTPUT	OUT(m),A	11 010 011 〈m〉	2	10	$Ar \to (Am)_I$ $m \to A_0 \sim A_7$ $Ar \to A_8 \sim A_{15}$	·	·	·	·	·	·
	OUT(C),g	11 101 101 01 g 001	3	10	$gr \to (BC)_I$ $Cr \to A_0 \sim A_7$ $Br \to A_8 \sim A_{15}$	·	·	·	·	·	·
	OUT0(m),g	11 101 101 00 g 001 〈m〉	3	13	$gr \to (00m)_I$ $m \to A_0 \sim A_7$ $00 \to A_8 \sim A_{15}$	·	·	·	·	·	·
	OTDM	11 101 101 10 001 011	2	14	$(HL)_M \to (00C)_I$ $HL_R - 1 \to HL_R$ $Cr - 1 \to Cr$ $Br - 1 \to Br$ $Cr \to A_0 \sim A_7$ $00 \to A_8 \sim A_{15}$	↕	⑤ ↕	↕	P	⑥ ↕	↕
	OTDMR	11 101 101 10 011 011	2	16(Br≠0) 14(Br=0)	Q($(HL)_M \to (00C)_I$ $HL_R - 1 \to HL_R$ $Cr - 1 \to Cr$ $Br - 1 \to Br$) Repeat Q until Br=0 $Cr \to A_0 \sim A_7$ $00 \to A_8 \sim A_{15}$	R	S	R	S	⑥ ↕	R
	OTDR	11 101 101 10 111 011	2	14(Br≠0) 12(Br=0)	Q($(HL)_M \to (BC)_I$ $HL_R - 1 \to HL_R$ $Br - 1 \to Br$) Repeat Q until Br=0 $Cr \to A_0 \sim A_7$ $Br \to A_8 \sim A_{15}$	X	S	X	X	⑥ ↕	X
	OUTI	11 101 101 10 100 011	2	12	$(HL)_M \to (BC)_I$ $HL_R + 1 \to HL_R$ $Br - 1 \to Br$ $Cr \to A_0 \sim A_7$	X	⑤ ↕	X	X	⑥ ↕	X

⑤ Z=1：Br−1=0
　Z=0：Br−1≠0

⑥ N=1：MSB of Data=1
　N=0：MSB of Data=0

（次頁に続く）

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
					Br→A_8～A_{15}						
	OTIR	11 101 101 10 110 011	2	14(Br≠0) 12(Br=0)	Q{ $(HL)_M$→$(BC)_I$ HL_R+1→HL_R Br−1→Br } Repeat Q until Br=0 Cr→A_0～A_7 Br→A_8～A_{15}	X	S	X	X	⑥ ↕	X
	TSTIO m	11 101 101 01 110 100 〈m〉	3	12	$(00C)_I$·m Cr→A_0～A_7 00→A_8～A_{15}	↕	↕	S	P	R	R
	OTIM	11 101 101 10 000 011	2	14	$(HL)_M$→$(00C)_I$ HL_R+1→HL_R Cr+1→Cr Br−1→Br Cr→A_0～A_7 00→A_8～A_{15}	↕	⑤ ↕	↕	P	⑥ ↕	↕
	OTIMR	11 101 101 10 010 011	2	16(Br≠0) 14(Br=0)	Q{ $(HL)_M$→$(00C)_I$ HL_R+1→HL_R Cr+1→Cr Br−1→Br } Repeat Q until Br=0 Cr→A_0～A_7 00→A_8～A_{15}	R	S	R	S	⑥ ↕	R
	OUTD	11 101 101 10 101 011	2	12	$(HL)_M$→$(BC)_I$ HL_R−1→HL_R Br−1→Br Cr→A_0～A_7 Br→A_8～A_{15}	X	⑤ ↕	X	X	⑥ ↕	X

⑤ Z=1：Br−1=0
　 Z=0：Br−1≠0

⑥ N=1：MSB of Data=1
　 N=0：MSB of Data=0

(5) 特殊制御命令

オペレーション名	ニーモニック	オペコード	バイト数	ステート数	オペレーション	フラグ					
						7	6	4	2	1	0
						S	Z	H	P/V	N	C
Special Function	DAA	00 100 111	1	4	Decimal Adjust Accumulator	↕	↕	↕	P	·	↕
Carry Control	CCF	00 111 111	1	3	$\bar{C}$→C	·	·	R	·	R	↕
	SCF	00 110 111	1	3	1→C	·	·	R	·	R	S
CPU Control	DI	11 110 011	1	3	0→IEF_1, 0→IEF_2 ⑦	·	·	·	·	·	·
	EI	11 111 011	1	3	1→IEF_1, 1→IEF_2 ⑦	·	·	·	·	·	·
	HALT	01 110 110	1	3	CPU halted	·	·	·	·	·	·
	IM 0	11 101 101 01 000 110	2	6	Interrupt mode 0	·	·	·	·	·	·
	IM 1	11 101 101 01 010 110	2	6	Interrupt mode 1	·	·	·	·	·	·
	IM 2	11 101 101 01 011 110	2	6	Interrupt mode 2	·	·	·	·	·	·
	NOP	00 000 000	1	3	No operation	·	·	·	·	·	·
	SLP	11 101 101 01 110 110	2	8	Sleep	·	·	·	·	·	·

⑦ DIまたはEI命令の最後では，割込みはサンプルされません．

2 内蔵I/Oレジスタ一覧

I/O レジスタのアドレスは，上位8ビットすべて“0”であり，下位8ビット中MSBから2ビットは，I/O コントロールレジスタ内の IOA7 と IOA6 により設定できます．以下の表に示すアドレスは，IOA7 と IOA6 が“0”の場合です．

レジスタ	アドレス	備考
ASCIコントロールレジスタ A チャネル0：CNTLA0	00	

ビット	MPE	RE	TE	$\overline{RTS0}$	MPBR/EFR	MOD2	MOD1	MOD0
リセット時	0	0	0	1	不定	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

レジスタ	アドレス	備考
ASCIコントロールレジスタA チャネル1：CNTLA1	01	

ビット	MPE	RE	TE	CKA1D	MPBR/EFR	MOD2	MOD1	MOD0
リセット時	0	0	0	0	不定	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

MOD2, 1, 0

MOD2	MOD1	MOD0	
0	0	0	Start＋7bit Data＋1stop
0	0	1	Start＋7bit Data＋2stop
0	1	0	Start＋7bit Data＋Parity＋1stop
0	1	1	Start＋7bit Data＋Parity＋2stop
1	0	0	Start＋8bit Data＋1stop
1	0	1	Start＋8bit Data＋2stop
1	1	0	Start＋8bit Data＋Parity＋1stop
1	1	1	Start＋8bit Data＋Parity＋2stop

レジスタ	アドレス	備考
ASCIコントロールレジスタB チャネル0：CNTLB0	02	(下記参照)
ASCIコントロールレジスタB チャネル1：CNTLB1	03	(下記参照)

ASCIコントロールレジスタB チャネル0：CNTLB0（アドレス 02）

ビット	MPBT	MP	$\overline{CTS}$/PS	PEO	DR	SS2	SS1	SS0
リセット時	不定	0	*	0	0	1	1	1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- SS2, SS1, SS0: Clock Source and Speed Select
- DR: Divide Ratio
- PEO: Parity Even or Odd
- $\overline{CTS}$/PS: Clear To Send/Prescale
- MP: Multi Processor
- MPBT: Multi Processor Bit Transmit

* 端子の状態を取り込みます

ASCIコントロールレジスタB チャネル1：CNTLB1（アドレス 03）

ビット	MPBT	MP	$\overline{CTS}$/PS	PEO	DR	SS2	SS1	SS0
リセット時	不定	0	0	0	0	1	1	1
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

- SS2, SS1, SS0: Clock Source and Speed Select
- DR: Divide Ratio
- PEO: Parity Even or Odd
- $\overline{CTS}$/PS: Clear To Send/Prescale
- MP: Multi Processor
- MPBT: Multi Processor Bit Transmit

総合分周比	PS=0（分周比=10）		PS=1（分周比=30）	
SS 2, 1, 0	DR=0(×16)	DR=1(×64)	DR=0(×16)	DR=1(×64)
0 0 0	φ÷ 160	φ÷ 640	φ÷ 480	φ÷ 1 920
0 0 1	÷ 320	÷ 1 280	÷ 960	÷ 3 840
0 1 0	÷ 640	÷ 2 560	÷ 1 920	÷ 7 680
0 1 1	÷ 1 280	÷ 5 120	÷ 3 840	÷ 15 360
1 0 0	÷ 2 560	÷10 240	÷ 7 680	÷ 30 720
1 0 1	÷ 5 120	÷20 480	÷15 360	÷ 61 440
1 1 0	÷10 240	÷40 960	÷30 720	÷122 880
1 1 1	外部クロック入力（÷40以上）			

レジスタ	アドレス	備考
ASCIステータスレジスタチャネル0：STAT0	04	（下記）
ASCIステータスレジスタチャネル1：STAT1	05	（下記）
ASCIトランスミットデータレジスタチャネル0：TDR0	06	
ASCIトランスミットデータレジスタチャネル1：TDR1	07	

STAT0（04）

ビット	RDRF	OVRN	PE	FE	RIE	$\overline{DCD0}$	TDRE	TIE
リセット時	0	0	0	0	0	*	**	0
R/W	R	R	R	R	R/W	R	R	R/W

*端子の状態を取り込みます

**

$\overline{CTS_0}$ 端子	TDRE
L	1
H	0

STAT1（05）

ビット	RDRF	OVRN	PE	FE	RIE	CTS1E	TDRE	TIE
リセット時	0	0	0	0	0	0	1	0
R/W	R	R	R	R	R/W	R/W	R	R/W

レジスタ	アドレス	備　　考
ASCIレシーブ データレジスタ チャネル0：RDR0	08	
ASCIレシーブ データレジスタ チャネル1：RDR1	09	
CSI/Oコントロールレジスタ：CNTR	0A	(see below)
CSI/Oトランスミット/レシーブデータレジスタ：TRDR	0B	
タイマデータ レジスタ チャネル0L：TMDR0L	0C	
タイマデータ レジスタ チャネル0H：TMDR0H	0D	
タイマリロード レジスタ チャネル0L：RLDR0L	0E	
タイマリロード レジスタ チャネル0H：RLDR0H	0F	

CNTR (0A):

ビット	EF	EIE	RE	TE	—	SS2	SS1	SS0
リセット時	0	0	0	0	1	1	1	1
R/W	R	R/W	R/W	R/W		R/W	R/W	R/W

- EF: End Flag
- EIE: End Interrupt Enable
- RE: Receive Enable
- TE: Transmit Enable
- SS2, SS1, SS0: Speed Select

SS 2, 1, 0	ボーレート	SS 2, 1, 0	ボーレート
0 0 0	φ÷ 20	1 0 0	φ÷ 320
0 0 1	÷ 40	1 0 1	÷ 640
0 1 0	÷ 80	1 1 0	÷1 280
0 1 1	÷160	1 1 1	外部(÷20以上)

<table>
<tr><th>レジスタ</th><th>アドレス</th><th>備　　考</th></tr>
<tr><td>タイマコントロールレジスタ：TCR</td><td>10</td><td>

ビット	TIF1	TIF0	TIE1	TIE0	TOC1	TOC0	TDE1	TDE0
リセット時	0	0	0	0	0	0	0	0
R/W	R	R	R/W	R/W	R/W	R/W	R/W	R/W

TDE1, TDE0: Timer Down Count Enable 1, 0
TOC1, TOC0: Timer Output Control 1, 0
TIE1, TIE0: Timer Interrupt Enable 1, 0
TIF1, TIF0: Timer Interrupt Flag 1, 0

TOC 1, 0	A_{18}/TOUT端子
0 0	タイマ出力禁止
0 1	トグル出力
1 0	"0" を出力
1 1	"1" を出力

</td></tr>
<tr><td>タイマデータレジスタチャネル1L：TMDR1L</td><td>14</td><td></td></tr>
<tr><td>タイマデータレジスタチャネル1H：TMDR1H</td><td>15</td><td></td></tr>
<tr><td>タイマリロードレジスタチャネル1L：RLDR1L</td><td>16</td><td></td></tr>
<tr><td>タイマリロードレジスタチャネル1H：RLDR1H</td><td>17</td><td></td></tr>
<tr><td>フリーランニングカウンタ：FRC</td><td>18</td><td>リードのみ</td></tr>
<tr><td>DMAソースアドレスレジスタチャネル0L：SAR0L</td><td>20</td><td></td></tr>
</table>

<table>
<tr><th>レジスタ</th><th>アドレス</th><th>備　考</th></tr>
<tr><td>DMAソース
アドレスレジスタ
チャネル0H：
SAR0H</td><td>21</td><td></td></tr>
<tr><td>DMAソース
アドレスレジスタ
チャネル0B：
SAR0B</td><td>22</td><td>ビット0, 1, 2(, 3*)のみ使用
<table>
<tr><th>A_{19}*, A_{18}, A_{17}, A_{16}</th><th></th><th></th><th></th><th>DMA 転送要求</th></tr>
<tr><td>×</td><td>×</td><td>0</td><td>0</td><td>$\overline{DREQ_0}$（外部）</td></tr>
<tr><td>×</td><td>×</td><td>0</td><td>1</td><td>RDR0　(ASCI0)</td></tr>
<tr><td>×</td><td>×</td><td>1</td><td>0</td><td>RDR1　(ASCI1)</td></tr>
<tr><td>×</td><td>×</td><td>1</td><td>1</td><td>Not Used</td></tr>
</table></td></tr>
<tr><td>DMAディスティネーションアドレスレジスタチャネル0L：DAR0L</td><td>23</td><td></td></tr>
<tr><td>DMAディスティネーションアドレスレジスタチャネル0H：DAR0H</td><td>24</td><td></td></tr>
<tr><td>DMAディスティネーションアドレスレジスタチャネル0B：DAR0B</td><td>25</td><td>ビット0, 1, 2(, 3*)のみ使用
<table>
<tr><th>A_{19}*, A_{18}, A_{17}, A_{16}</th><th></th><th></th><th></th><th>DMA 転送要求</th></tr>
<tr><td>×</td><td>×</td><td>0</td><td>0</td><td>$\overline{DREQ_0}$（外部）</td></tr>
<tr><td>×</td><td>×</td><td>0</td><td>1</td><td>TDR0　(ASCI0)</td></tr>
<tr><td>×</td><td>×</td><td>1</td><td>0</td><td>TDR1　(ASCI1)</td></tr>
<tr><td>×</td><td>×</td><td>1</td><td>1</td><td>Not Used</td></tr>
</table></td></tr>
<tr><td>DMAバイト
カウントレジスタ
チャネル0L：
BCR0L</td><td>26</td><td></td></tr>
<tr><td>DMAバイト
カウントレジスタ
チャネル0H：
BCR0H</td><td>27</td><td></td></tr>
<tr><td>DMAメモリ
アドレスレジスタ
チャネル1L：
MAR1L</td><td>28</td><td></td></tr>
<tr><td>DMAメモリ
アドレスレジスタ
チャネル1H：
MAR1H</td><td>29</td><td></td></tr>
</table>

＊　CP-68, FP-80のみ有効

レジスタ	アドレス	備考
DMAメモリアドレスレジスタ チャネル1B：MAR1B	2A	ビット0、1、2（、3*）のみ使用
DMAI/Oアドレスレジスタ チャネル1L：IAR1L	2B	
DMAI/Oアドレスレジスタ チャネル1H：IAR1H	2C	
DMAバイトカウントレジスタ チャネル1L：BCR1L	2E	
DMAバイトカウントレジスタ チャネル1H：BCR1H	2F	
DMAステータスレジスタ：DSTAT	30	（下表参照）
DMAモードレジスタ：DMODE	31	（下表参照）

DMAステータスレジスタ：DSTAT（30）

ビット	DE1	DE0	$\overline{DWE1}$	$\overline{DWE0}$	DIE1	DIE0	—	DME
リセット時	0	0	1	1	0	0	1	0
R/W	R/W	R/W	W	W	R/W	R/W		R

DME：DMA Master Enable
DIE1, DIE0：DMA Interrupt Enable 1、0
DWE1, DWE0：DMA Enable Bit Write Enabl 1、0
DE1, DE0：DMA Enable ch 1、0

DMAモードレジスタ：DMODE（31）

ビット	—	—	DM1	DM0	SM1	SM0	MMOD	—
リセット時	1	1	0	0	0	0	0	1
R/W			R/W	R/W	R/W	R/W	R/W	

MMOD：Memory MODE Select
SM1, SM0：ch 0 Source Mode 1、0
DM1, DM0：ch 0 Destination Mode 1、0

<table>
<tr><th>レジスタ</th><th>アドレス</th><th>備　　考</th></tr>
<tr><td></td><td></td><td>

DM 1, 0	ディスティネーション	アドレス
0 0	M	DAR0+1
0 1	M	DAR0−1
1 0	M	DAR0固定
1 1	I/O	DAR0固定

SM 1, 0	ソース	アドレス
0 0	M	SAR0+1
0 1	M	SAR0−1
1 0	M	SAR0固定
1 1	I/O	SAR0固定

MMOD	モード
0	サイクルスチールモード
1	バーストモード

</td></tr>
<tr><td>DMA/WAIT
コントロール
レジスタ：DCNTL</td><td>32</td><td>

ビット	MWI1	MWI0	IWI1	IWI0	DMS1	DMS0	DIM1	DIM0
リセット時	1	1	1	1	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

MWI1, MWI0: Memory Wait Insertion
IWI1, IWI0: I/O Wait Insertion
DMS1, DMS0: $\overline{DREQi}$ Select, i=1, 0
DIM1, DIM0: DMA ch 1 I/O Memory Mode Select

MWI 1, 0	ウェイトステート
0 0	0
0 1	1
1 0	2
1 1	3

IWI 1, 0	ウェイトステート
0 0	0
0 1	2
1 0	3
1 1	4

DMSi	$\overline{DREQi}$入力
1	エッジ入力
0	レベル入力

DIM 1, 0	転送形態	アドレス増減	
0 0	M→I/O	MAR1+1	IAR1固定
0 1	M→I/O	MAR1−1	IAR1固定
1 0	I/O→M	IAR1固定	MAR1+1
1 1	I/O→M	IAR1固定	MAR1−1

</td></tr>
<tr><td>インタラプトベクタローレジスタ：IL</td><td>33</td><td>

ビット	IL7	IL6	IL5	—	—	—	—	—
リセット時	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W					

IL7–IL5: Interrupt Vector Low

</td></tr>
</table>

<table>
<tr><th>レジスタ</th><th>アドレス</th><th>備　　考</th></tr>
<tr><td>INT/TRAR
コントロール
レジスタ：ITC</td><td>34</td><td>

ビット	TRAP	UFO	—	—	—	ITE2	ITE1	ITE0
リセット時	0	0	1	1	1	0	0	1
R/W	R/W	R				R/W	R/W	R/W

TRAP: TRAP
UFO: Undefined Fetch Object Code
ITE2, ITE1, ITE0: $\overline{INT}$ Enable 2、1、0
</td></tr>
<tr><td>リフレッシュ
コントロール
レジスタ：RCR</td><td>36</td><td>

ビット	REFE	REFW	—	—	—	—	CYC1	CYC0
リセット時	1	1	1	1	1	1	0	0
R/W	R/W	R/W					R/W	R/W

REFE: Refresh Enable
REFW: Refresh Wait State
CYC1, CYC0: Cycle Select

CYC 1、0	リフレッシュサイクル間隔
0 0	10ステート
0 1	20
1 0	40
1 1	80

</td></tr>
<tr><td>MMUコモン
ベースレジスタ：
CBR</td><td>38</td><td>

ビット	CB7*	CB6	CB5	CB4	CB3	CB2	CB1	CB0
リセット時	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

CB7～CB0: MMU Common Base Register
</td></tr>
<tr><td>MMUバンク
ベースレジスタ：
BBR</td><td>39</td><td>

ビット	BB7*	BB6	BB5	BB4	BB3	BB2	BB1	BB0
リセット時	0	0	0	0	0	0	0	0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

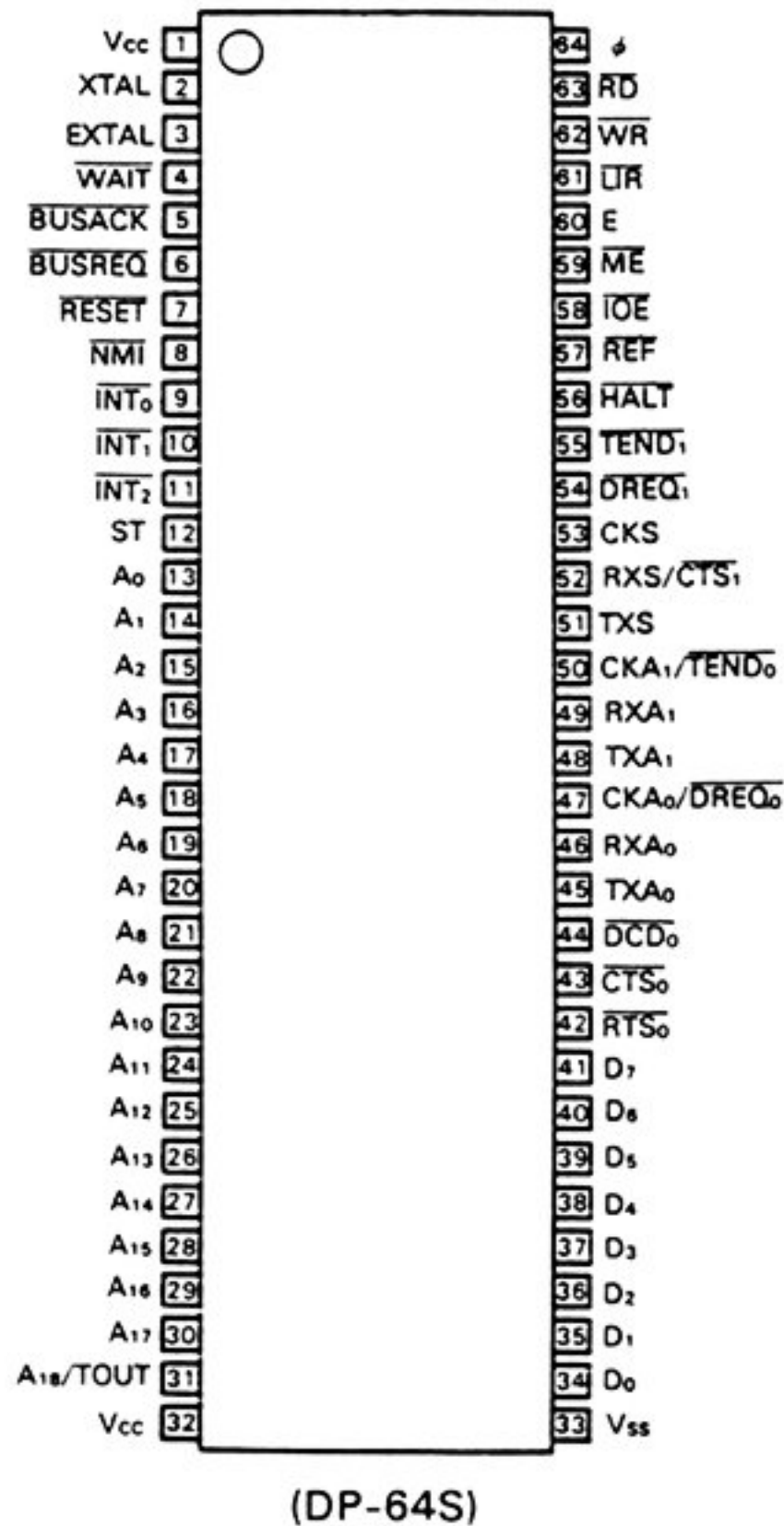
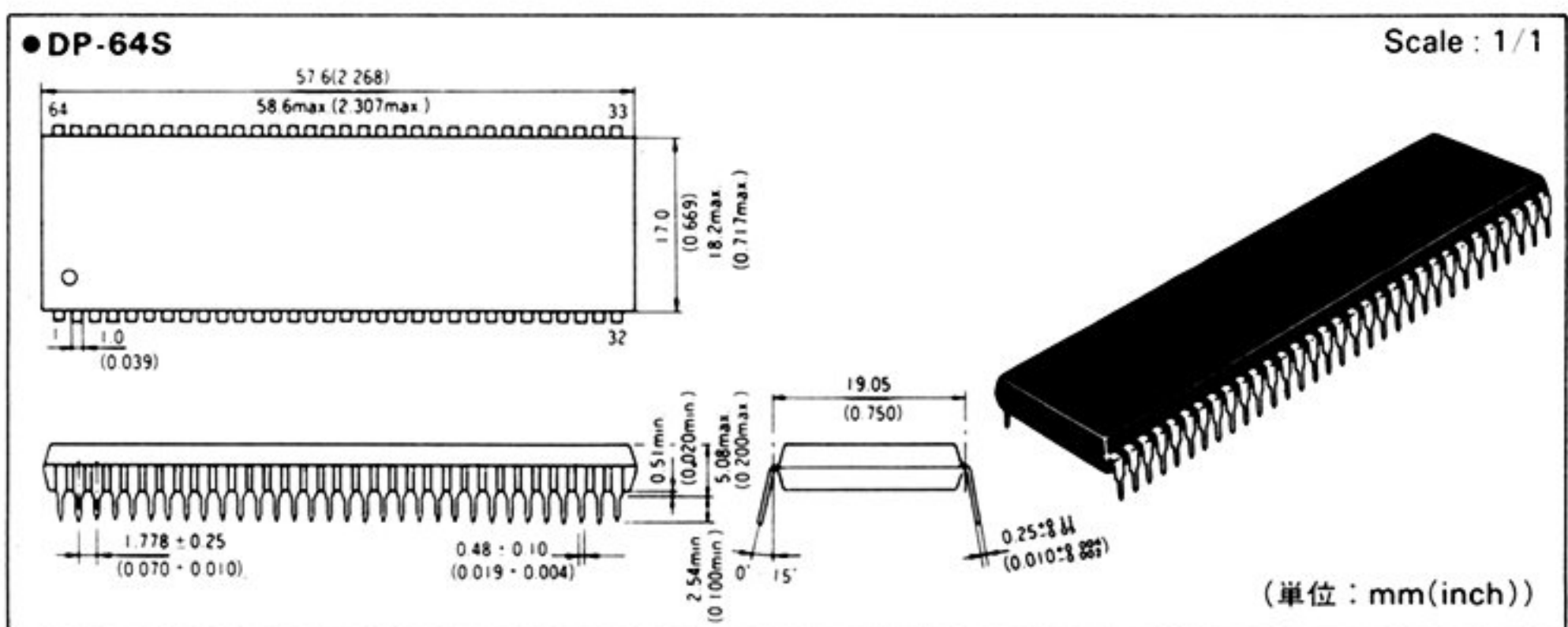
BB7～BB0: MMU Bank Base Register
</td></tr>
</table>

＊　CP-68，FP-80のみ有効

レジスタ	アドレス	備考								
MMUコモン/バンクエリアレジスタ：CBAR	3A	ビット	CA3	CA2	CA1	CA0	BA3	BA2	BA1	BA0
		リセット時	1	1	1	1	0	0	0	0
		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
			MMU Common Area Register				MMU Bank Area Register			
動作モードコントロールレジスタ：OMCR（Z版のみ有効）	3E	ビット	LIRE	$\overline{\text{LIRTE}}$	$\overline{\text{IOC}}$	—	—	—	—	—
		リセット時	1	1	1	1	1	1	1	1
		R/W	R/W	W	R/W					
			LIR Enable	$\overline{\text{LIR}}$ Temporary Enable	I/O Compatibility					
I/Oコントロールレジスタ：ICR	3F	ビット	IOA7	IOA6	IOSTP	—	—	—	—	—
		リセット時	0	0	0	1	1	1	1	1
		R/W	R/W	R/W	R/W					
			I/O Address		I/O Stop					

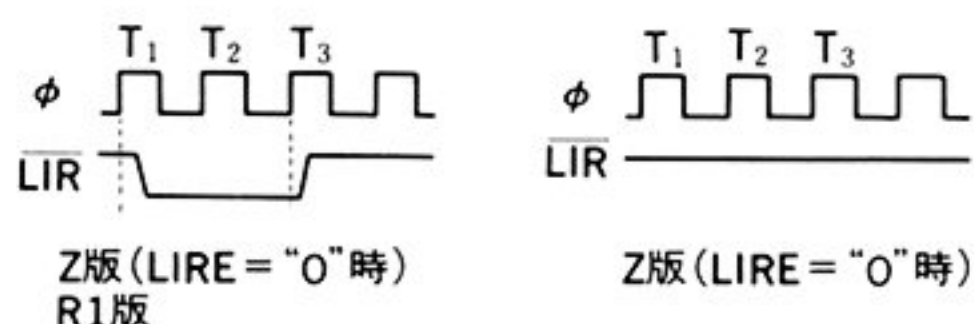
3　端子名一覧

1. ピン配置図（上面図）

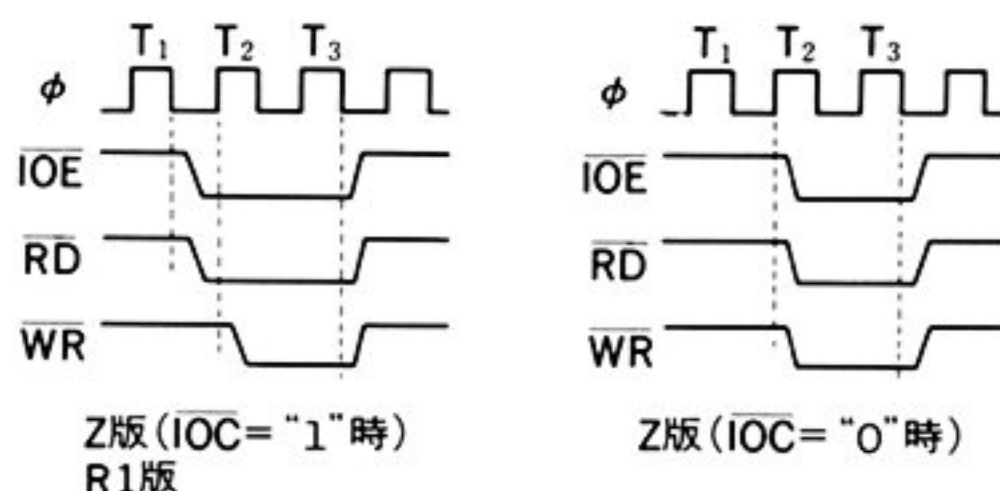
(DP-64S)

2. 外 形 寸 法 図

4　R1版とZ版との違い

64180Z版は，64180R1版に対して上位互換性をもっています．Z版には次のような機能が追加されています．

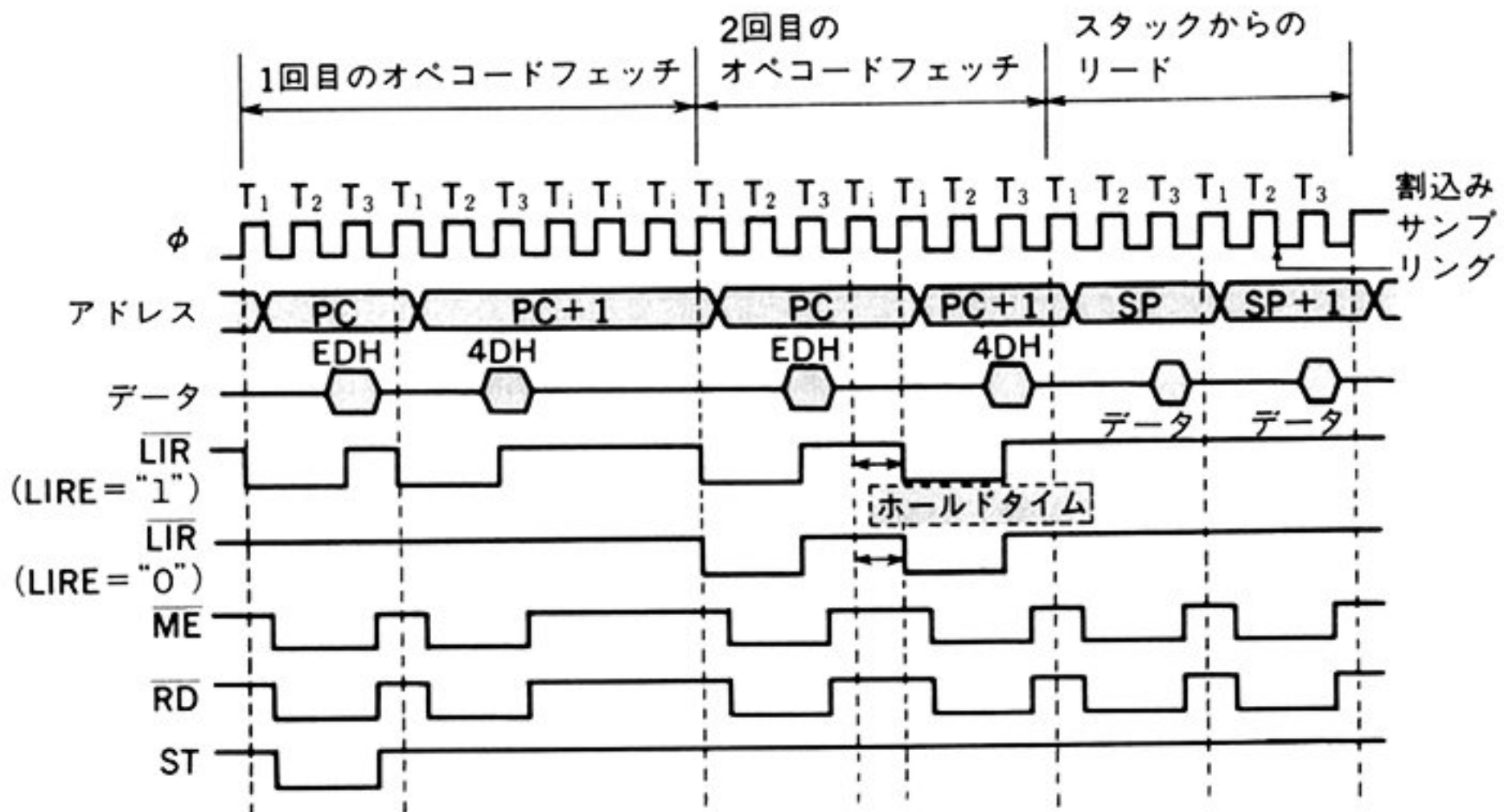
1.　$\overline{\text{LIR}}$ タイミング

付図1 に示すように，オペコードフェッチサイクルでも "H" のままにすることができます．動作モードコントロールレジスタ(OMCR)内のLIREビットに "0" を書き込んだ後の最初のオペコードフェッチから，この状態になります．この状態が必要になるのは，Z80周辺LSIをデイジーチェイン割込みで使用する場合です．

付図1　$\overline{\text{L±R}}$ タイミング

2.　$\overline{\text{IOE}}$，$\overline{\text{RD}}$ タイミング

付図2 に示すように，$\overline{\text{IOE}}$ と $\overline{\text{RD}}$ の出力タイミングがOMCR内の $\overline{\text{IOC}}$ ビットの状態に従って変わります．$\overline{\text{IOC}}$ ビットに "0" を書き込んだ場合のタイミングでは，Z80周辺LSIとのインタフェースが容易になります．

付図2　$\overline{\text{IOE}}$，$\overline{\text{RD}}$ タイミング

3.　RETI命令

この命令(コードはEDH, 4DH)は，Z80周辺LSIがデイジーチェイン割込み接続しているときに，割込み要求デバイスが内部の割込み要求をクリアするためにサポートされています．R1版では，この命令を解読するためのホールドタイムが十分ではありません．Z版ではこの点を改良し，**付図3** のように第1オペコードフェッチサイクルと第2オペコードフェッチサイクルに1ステートダミーサイク

R1版は2回目のオペコードフェッチはなく，かつLIRE="1"の状態に等しい．

付図 3　RETI 命令の実行タイミング（Z 版）

ル T_i を挿入し，解読のためのホールド時間をもたせています．ただし，CPUはこの命令を自動的に2度フェッチしているので，1回目のフェッチ時には $\overline{\text{LIR}}$ はインアクティブにします．この操作は，上記の OMCR の LIRE ビットへ"0"を書き込むことにより行います．付図3のように，2回目のフェッチ時のみ $\overline{\text{LIR}}$ がアクティブとなります．

なお，Z80PIO は内部の割込み回路をイネーブルとするために，$\overline{\text{M1}}$ 信号の立上りエッジを利用しています．ところが，Z版は LIRE ビットを"0"としたとき，$\overline{\text{LIR}}$ ($\overline{\text{M1}}$) が出力されません．そこで，$\overline{\text{LIR}}$ 信号を1回だけ出力させる目的で，Z版は OMCR 内に $\overline{\text{LIRTE}}$ ビットをもっています．$\overline{\text{LIR}}$ 信号は，$\overline{\text{LIRTE}}$ ビットへ"0"を書き込んだ後，最初に実行されるオペコードフェッチサイクルで出力されます．ただし，このオペコードフェッチ時のデータバスには，EDH，CBH，FDH，DDH が現れないようにします．すなわち，第2オペコードをもつ命令を実行させないようにします．その具体的なプログラミング例を**付図4**に示します．

```
DI                      ：割込み受付の禁止
PUSH AF                 ：アキュムレータの退避
INO  A,(OMCR)      ┐
RES  6,A           ├ ：LIRTE ビットに"0"を書き込む
OUTO (OMCR),A      ┘
POP  AF                 ：アキュムレータの内容を復帰
                         (この命令は1バイト命令)
EI                      ：割込み受付の許可
```

付図 4　$\overline{\text{LIRTE}}$ ビットへの"0"書込み例

5　180カードパソコンのハードウェア仕様

No.	項目		仕様	備考
1	プロセッサ（MPU）		HD64180R1P6(6MHz)	HITACHI
2	動作周波数		・φ＝6.144MHz ・外部水晶振動子……12.288MHz	
3	記憶容量（RAM）	使用メモリ	HM50256-15	
		使用数	8個(すべてデータメモリ)	
		総容量	256Kバイト	
	記憶容量（ROM）	使用メモリ	HN27256G-25またはHN27C64G-25 (HN27C256G-25)/HN：27128G-25	切替可能
		使用数	1個	
		総容量	32KB/16KB/8KB	
4	シリアルインタフェース		・RS-232C通信回線……2ch ……64180内蔵SCI使用 ・モデム制御信号($\overline{RTS}$, $\overline{CTS}$, $\overline{DCD}$)有り ……チャネル0のみ ・チャネル1はターミナル専用……転送レート：9600bps, ビット長：8ビット，パリティ：なし，ストップビット：2ストップビット	
5	フロッピーディスクインタフェース		・5インチ両面倍密 ・専用コントローラ使用……HD63，265 ・データセパレータ内蔵 ・使用ドライバ可能台数……最大4台まで	
		ディスクフォーマット	Fコマンドでフォーマット可能	モニタプログラム
		記録方式	MFM	
		データ転送	HD64180内蔵DMACによるDMA転送 （デュアルアドレス方式）	
		セクタ長	256バイト/セクタ	
		対象FDD	FD-55BV	TEAC
6	フロッピーディスクドライブ(外付)	データ容量　フォーマットなし	500Kバイト	
		データ容量　フォーマットあり	327.68Kバイト	256バイト/セクタ
		トラック密度	48tpi	
		シリンダ数	40	

No.	項目			仕様	備考
6	フロッピーディスクドライブ(外付)	トラック数		80	
		記録方式		MFM	
		ディスク回転速度		300 rpm	
		データ転送速度		250Kビット/秒	
		回転平均待ち時間		100 mS	
		アクセス時間	トラック時間	6 mS	
			セトリング時間	15 mS	
		ヘッド・ロード時間		35 mS	
		モータ起動時間		400 mS	
7	パラレルインタフェース			・セントロニクス（8ビットパラレル）インタフェース準拠 ・$\overline{\mathrm{ACK}}$ 信号応答によるハンドシェイク	
8	バス・サイクル数	メモリアクセス		T_1, T_2, T_W, T_3 の4サイクル	
		I/O アクセス		T_1, T_2, T_W(T_W, T_W, T_W) T_3	(　)はソフト設定
9	電源電圧および消費電流	電圧		5V ± 5%	
				±12V ± 5%	
		電流		450 mA (TYP)	
10	モニタ			・ROM に内蔵 ・11 種のコマンド	
11	リセット検出電圧			4.5V ± 3.3% (PST521C使用)	
12	リセット条件			パワーオンまたは電圧低下によるリセット検出回路動作時	リセットSW付
13	I/O端子			フラットケーブル用60Pコネクタによる接続 型式：FRC2-C60L11-OL(DDK)	
14	電源端子			4PIN (+5V, +12V, −12V, GND) コネクタによる接続 型式：IL-4P-S3EV2-1(JAE)	

参 考 文 献

・横田英一：図解マイクロコンピュータ Z-80 の使い方，オーム社（1981）

・HD64180 ハードウェアマニュアル，日立製作所

・COMPONENTS DATA BOOK，Zilog 社

・各社半導体規格表

索　　引

ア　行

カ　行

サ　行

タ　行

索　　　　　　　引

編 者 略 歴

石川　知雄（いしかわ　ともお）

昭和29年　東北大学工学部
　　　　　通信工学科卒業

昭和35年　スタンフォード大学
　　　　　大学院修了（Ph. D.）

現　　在　株式会社日立製作所

図解 マイクロコンピュータ

6 4 1 8 0 の 使 い 方

昭和63年 1 月 20日　　第1版第1刷発行

OHM・OHM・OHM・OHM
編者承認
検印省略

編　者　石　川　知　雄

発行者　株式会社 **オ ー ム** 社
代表者　種 田 則 一

発行所　株式会社 **オ ー ム** 社
郵便番号 101
東京都千代田区神田錦町3-1
振 替　東京 6-20018
電 話　03(233)0641(代表)

組版 エヌ・ピー・エス　印刷 日東プリテック　製本 協栄製本

Printed in Japan

落丁・乱丁本はお取替えいたします

ISBN 4 - 274 - 07396 - 3

図書案内

図解コンピュータシリーズ
江村潤朗 監修

カバー印刷：三和印刷